

Lambda-Free Logical Frameworks[☆]

Robin Adams^a

^a*Royal Holloway, University of London*

Abstract

We present the definition of the logical framework TF, the *Type Framework*. TF is a lambda-free logical framework; it does not include lambda-abstraction or product kinds. We give formal proofs of several results in the metatheory of TF, and show how it can be conservatively embedded in the logical framework LF: its judgements can be seen as the judgements of LF that are in beta-normal, eta-long normal form. We show how several properties, such as the injectivity of constants and the strong normalisation of an object theory, can be proven more easily in TF, and then ‘lifted’ to LF.

Key words: logical framework, type theory, lambda-free
2000 MSC: 03B15, 03B22, 03B35, 03B70, 68T15

1. Introduction

A *logical framework* is a typing system intended as a meta-language for the specification of other formal systems, which may themselves be type theories or other systems of logic, such as predicate logic. Traditionally, logical frameworks are based on a typed lambda calculus; variable binding is represented by lambda-abstraction in the framework, and substitution by application in the framework. The correspondence between the object theory and its representation in the framework is not exact: each entity of the object theory is represented by more than one object in the framework — typically, $\beta\eta$ -convertible objects represent the same entity of the object theory — and there are objects in the framework (such as partially applied meta-functions) that do not correspond to any entity of the object theory. It is therefore necessary to prove *adequacy theorems* establishing the relationship between an object theory and its representation in a logical framework; and these theorems are notoriously often difficult to prove.

[☆]Corresponding address: Department of Computer Science, Royal Holloway, University of London, Egham Hill, Egham, Surrey. TW20 0EX. England. Tel.: +44 1784 443421. Fax: +44 1784 439786. Email robin@cs.rhul.ac.uk
 This research was supported by the UK EPSRC research grant Pythagoras GR/R84092, the EU Framework VI grant TYPES 510996, and the UK EPSRC research fellowship EP/D066638/1.

It is possible to construct a logical framework that does not employ all the apparatus of the lambda calculus. We can construct logical frameworks that do not make use of abstraction and substitution, but instead involve only parametrisation and the instantiation of parameters. We shall call these *lambda-free* logical frameworks. They can be seen as frameworks that only use β -normal, η -long normal forms. Lambda-free frameworks provide a more faithful representation of an object theory — there is a one-to-one correspondence between the objects of the framework and the terms and types of the object theory. Because of this, many results including adequacy theorems are easier to prove in a lambda-free framework.

It is often possible to *embed* a lambda-free framework L within a traditional framework F ; that is, to provide a translation from L into F such that the derivable judgements of L map onto exactly the derivable judgements of F that are in normal form. F can then be seen as a conservative extension of L . Once this embedding has been established, we can ‘lift’ results from L to F ; that is, we can prove a result for L , and then deduce that the corresponding result holds for F as a corollary.

There is a price to be paid for using a lambda-free framework: the early metatheoretic results are much more difficult to establish, as is the soundness of the embeddings discussed above. But this is a ‘one-time’ cost; once this price has been paid, it is comparatively easy to prove many results in the lambda-free framework, and then lift them to the traditional frameworks.

1.1. Background and Outline

The term ‘lambda-free logical framework’ was first used to describe the framework PAL^+ [1], which uses parametrisation and definitions as its basic notions rather than lambda-abstraction. In PAL^+ , however, it is possible to form abstractions (using parametric definition) that can then be applied to objects.

We are using the phrase ‘lambda-free logical framework’ in a stricter sense, to describe a framework which does not permit abstractions to be applied to objects, and which therefore contains no framework-level notion of reduction. We shall use the phrase ‘traditional framework’ throughout this paper to denote a logical framework that is not lambda-free, such as the Edinburgh LF [2] or Martin-Löf’s Logical Framework [3]. When we represent a formal system S within a logical framework F , the system S is referred to as the *object theory*.

The framework TF first appeared in an unpublished note by Aczel [4]. It was developed by myself in my thesis [5]. In particular, I introduced the set of arities to organise the grammar, and made explicit the definition of instantiation.

In Section 2, we give the formal definition of TF, and describe how a type theory may be specified in TF. In Section 3, we begin to prove the metatheoretic properties of TF. We would like to prove that these properties hold under an arbitrary type theory specification in TF. However, for most of the properties considered in Section 3, we are at present only able to prove them for two large classes of specifications — those with no equation declarations, and those which

do not involve variables of order 2 or higher¹. The proofs are given in Section 3, with the more technical proofs given in the Appendix.

In Section 4, we describe a second lambda-free logical framework TF_k , which is a Church-typed version of TF; that is, the bound variables are labelled with their kinds. We define translations between TF and TF_k in Section 4. It is often very convenient to have these two versions of TF available, and to be able to move between them at will.

In Section 5, we show how TF may be embedded in LF, a Church-typed version of Martin-Löf’s Logical Framework [6]. We do so by defining a translation from TF_k to LF and from LF to TF, taking advantage of the results of Section 4. We show how this embedding allows results to be *lifted*; that is, a result may be proven to hold for TF, and the fact that it holds for LF follows as an easy corollary. We demonstrate this for two results: the injectivity of type constructors, and strong normalisation of an object theory.

In Section 6, we describe two other frameworks that have appeared in the literature which are lambda-free logical frameworks in the stricter sense: the Concurrent Logical Framework (Concurrent LF) [7, 8] and DMBEL [9, 10]. In both of these frameworks, abstractions may be formed, and a constant or variable may be applied to an abstraction, but abstractions may not themselves be applied to objects.

Each of these may be conservatively embedded in TF. That is, we can find a subsystem S of TF such that there exist bijective translations between Concurrent LF and S , and such that TF is a conservative extension of S . Likewise, we can find a subsystem S' such that there exist bijective translations between DMBEL and S' , and such that TF is a conservative extension of S' . It is possible to find many such subsystems of TF, which all extend one another conservatively; this idea, called a ‘modular hierarchy of logical frameworks’, was described in Adams [11] and the formal details given in Adams [5]. We give the details in the case of Concurrent LF and DMBEL in Section 6.

Abbreviation. Throughout this paper, the phrase ‘induction hypothesis’ shall be abbreviated to ‘i.h.’.

2. The Type Framework TF

We present our first example of a lambda-free framework, the *Type Framework* TF. The framework TF includes nothing but what is essential for representing an object theory. In particular, it contains neither lambda-abstraction nor local definition; its basic concepts are parametrisation, the instantiation of parameters, and the declaration of equations.

¹In Adams [5], the properties in Section 3 were claimed to hold under an arbitrary specification, but a mistake has since been found in the proof.

2.1. Grammar

2.1.1. Arities

We begin by introducing the set of *arities*, with which we shall organise the syntax of TF.

The arities are defined inductively thus:

If $\alpha_1, \dots, \alpha_n$ are arities, then $(\alpha_1, \dots, \alpha_n)$ is an arity.

The base case of this definition is the case $n = 0$, yielding the arity $()$, which we shall write as $\mathbf{0}$. The next arities that can be formed are

$$\overbrace{(\mathbf{0}, \dots, \mathbf{0})}^n$$

for positive n ; we shall write this arity as \mathbf{n} . The next arities that can be formed are $(\mathbf{n}_1, \dots, \mathbf{n}_k)$, and so forth.

The intuition behind the arities is that an $(\alpha_1, \dots, \alpha_n)$ -ary function is a function that takes n arguments — namely an α_1 -ary function, \dots , and an α_n -ary function — and returns an entity (term or type) of the object theory. In particular, a $\mathbf{0}$ -ary (or *base*) function is just an entity of the object theory; a $\mathbf{2}$ -ary function is a binary operation on the entities of the object theory; and so forth.

We denote by $\alpha \hat{\ } \beta$ the *concatenation* of the two arities α and β :

$$(\alpha_1, \dots, \alpha_m) \hat{\ } (\beta_1, \dots, \beta_n) \equiv (\alpha_1, \dots, \alpha_m, \beta_1, \dots, \beta_n) \ .$$

We also ascribe an *order* to each arity as follows:

- The only 0th-order, or *base*, arity is $\mathbf{0}$.
- If the highest order among the arities $\alpha_1, \dots, \alpha_n$ is k , then $(\alpha_1, \dots, \alpha_n)$ is a $k + 1$ st-order arity.

For example, the first-order arities are those of the form \mathbf{n} for positive n , and the second-order arities are those of the form $(\mathbf{n}_1, \dots, \mathbf{n}_k)$ where at least one n_i is positive.

We say the arity α is a *subarity* of the arity β if α occurs inside β . We say α is a *proper* subarity of β if α is a subarity of β and $\alpha \neq \beta$.

2.2. Objects

The *objects* of TF are expressions intended to represent the terms and types of the object theory. They are built up from *variables* and *constants*, to each of which is assigned an arity. The constants shall be used for the type constructors and term constructors of the object theory. The variables shall be used as the variables of the object theory.

The set of objects is defined by the following inductive definition:

If z is an α -ary constant or variable, where

$$\alpha \equiv ((\alpha_{11}, \dots, \alpha_{1r_1}), \dots, (\alpha_{n1}, \dots, \alpha_{nr_n})) ,$$

then

$$z[[x_{11}, \dots, x_{1r_1}]M_1, \dots, [x_{n1}, \dots, x_{nr_n}]M_n] \quad (1)$$

is an object, where each x_{ij} is an α_{ij} -ary variable, and each M_i an object. Each x_{ij} is bound within the corresponding object M_i , and we identify objects up to α -conversion.

The base case of this definition is that, if z is a base variable or constant (that is, a $\mathbf{0}$ -ary variable or constant), then $z[]$ is an object; we shall henceforth write this object as just z . Likewise, if z is an \mathbf{n} -ary variable or constant, then $z[[M_1, \dots, [M_n]]$ is an object for any objects M_1, \dots, M_n ; we shall write this object simply as $z[M_1, \dots, M_n]$.

The subexpressions of the object (1) such as $[x_1, \dots, x_r]M$ are not first-class entities of TF; they cannot occur except as arguments to some variable or constant z . Nevertheless, it shall be convenient to have some way of referring to these pieces of raw syntax. We shall therefore introduce the following terminology:

- An $(\alpha_1, \dots, \alpha_n)$ -ary *variable sequence* is a sequence of n distinct variables $\langle x_1, \dots, x_n \rangle$, where x_i has arity α_i .
- An α -ary *abstraction* is an expression of the form $[\vec{x}]M$, where \vec{x} is an α -ary variable sequence, and M an object. We take each member of \vec{x} to be bound within this abstraction, and identify abstractions up to α -conversion.
- An $(\alpha_1, \dots, \alpha_n)$ -ary *abstraction sequence* is a sequence $\langle F_1, \dots, F_n \rangle$, where F_i is an α_i -ary abstraction.

Thus, an object has the form $z[\vec{F}]$, where z is an α -ary variable or constant, and \vec{F} an α -ary abstraction sequence. We shall often write this object as just $z\vec{F}$.

We note that the only expressions that can occur as arguments to a symbol are abstractions. In the situations where we would naturally wish to write a variable or constant in an argument position, we instead write its η -long form.

Definition2.1 (η -long Form). Given any α -ary variable or constant z , the η -long form z^η of z is the α -ary abstraction defined by recursion on α as follows:

If $\alpha \equiv (\alpha_1, \dots, \alpha_n)$, then

$$z^\eta \equiv [x_1, \dots, x_n]z[x_1^\eta, \dots, x_n^\eta] ,$$

where each x_i is an α_i -ary variable. (By α -conversion, it does not matter which variables we choose.)

2.3. Hereditary Substitution and Employment

We cannot use the familiar operation of substitution in TF. The result of substituting an abstraction $[\vec{y}]M$ for the variable x in the object $x\vec{F}$ is not an object of TF; rather, it would be a β -redex.

Instead, we introduce an operation that we name *instantiation*. The operation of *instantiating* an abstraction F for a variable x can be thought of as substituting F for x , then *reducing to normal form* (that is, β -normal, η -long form). However, we note that the definition does not use any notion of reduction.

Definition2.2 (Instantiation). Given an α -ary abstraction F , an α -ary variable x , and an object N , the object $\{F/x\}N$, the result of *instantiating* F for x in N , is defined by recursion firstly on the arity α , secondly on the object N , as follows:

$$\{F/x\}z[G_1, \dots, G_n] \equiv z[\{F/x\}G_1, \dots, \{F/x\}G_n] \quad (z \neq x)$$

If $F \equiv [t_1, \dots, t_n]P$, then

$$\{F/x\}x[G_1, \dots, G_n] \equiv \{\{F/x\}G_1/t_1\} \cdots \{\{F/x\}G_n/t_n\}P \ .$$

We assume here, through α -conversion, that no t_i occurs free in any G_j .

We shall also introduce a notational convention that shall play the role of abstraction: if x is an α -ary variable and F a β -ary abstraction, then $[x]F$ is an $(\alpha)\hat{\beta}$ -ary abstraction, defined by

$$[x][y_1, \dots, y_n]M \equiv [x, y_1, \dots, y_n]M \ .$$

Finally, we define an operation, which we shall call *employment*, to play the role usually taken by application. The result of *employing* F on G , denoted $F \bullet G$, can be thought of as the normal form of the application FG . The definition is:

Definition2.3 (Employment). Given an $(\alpha)\hat{\beta}$ -ary abstraction $[x]F$ and an α -ary abstraction G , the β -ary abstraction $F \bullet G$, the result of *employing* $[x]F$ on G , is defined by

$$([x]F) \bullet G \equiv \{G/x\}F \ .$$

We have used our newly introduced notation $[x]M$ in this definition; written out in full, the above equation is

$$([x, y_1, \dots, y_n]M) \bullet G \equiv [y_1, \dots, y_n]\{G/x\}M \ .$$

We shall abbreviate the repeated use of employment as follows: if \vec{G} is the abstraction sequence $\langle G_1, \dots, G_n \rangle$, then $F \bullet \vec{G}$ abbreviates $F \bullet G_1 \bullet G_2 \bullet \cdots \bullet G_n$, that is,

$$((\cdots (F \bullet G_1) \bullet G_2) \bullet \cdots) \bullet G_n \ .$$

Remark. We note that there is a strong correspondence between our syntax and the simply-typed lambda calculus. Our arities correspond to the types of the simply-typed lambda calculus, and our abstractions to the terms. Instantiation corresponds to the strategy of innermost reduction. Thus, the fact that our definition of instantiation is total corresponds to the fact that the simply-typed lambda calculus is weakly normalisable.

2.4. Kinds

A *base kind* in TF is either the symbol **Type**, or has the form $\text{El}(A)$ for some object A . The intention is that each type T of the object theory is represented by an object $\llbracket T \rrbracket$ of kind **Type**; the terms of type T are then represented by the objects of kind $\text{El}(\llbracket T \rrbracket)$.

In addition to these, we introduce a set of α -ary *product kinds* for every arity α . These shall be used to give kinds to the variables and constants of higher arity. The definition is by recursion on α :

An $(\alpha_1, \dots, \alpha_n)$ -ary product kind is an expression of the form

$$(x_1 : K_1, \dots, x_n : K_n)T \quad (2)$$

where the x_i s are distinct variables, x_i being of arity α_i ; each K_i is an α_i -ary product kind; and T is a base kind.

We take each variable x_i to be bound within $K_{i+1}, K_{i+2}, \dots, K_n$ and T in this product kind, and identify product kinds up to α -conversion.

The intuition is that the kind (2) represents the collection of functions that take n arguments — namely F_1 of kind K_1 , F_2 of kind $\{F_1/x_1\}K_2, \dots$, and F_n of kind $\{F_1/x_1, \dots, F_{n-1}/x_{n-1}\}K_n$ — and returns an object of kind $\{F_1/x_1, \dots, F_n/x_n\}T$.

If $K \equiv (x_1 : K_1, \dots, x_n : K_n)T$, then we shall write $(y : J)K$ for $(y : J, x_1 : K_1, \dots, x_n : K_n)T$.

Just as with abstractions, so the product kinds of non-zero arity are not considered first-class entities of TF; only the base kinds are. We shall however make use of the higher product kinds to give kinds to the variables and constants of higher arity. We shall even talk of an abstraction being a member of a product kind; however, this shall not be represented by a primitive judgement form of TF.

Contexts. A *context* Γ in TF is a sequence of the form:

$$x_1 : K_1, \dots, x_n : K_n$$

where the x_i s are distinct variables, and each x_i has the same arity as the corresponding product kind K_i . If each x_i has arity α_i , we say the context Γ has arity $(\alpha_1, \dots, \alpha_n)$, and its order $o(\Gamma)$ is then the order of $(\alpha_1, \dots, \alpha_n)$. The variable sequence $\langle x_1, \dots, x_n \rangle$ is called the *domain* of the context Γ , $\text{dom } \Gamma$.

Thus, an α -ary kind has the form $(\Gamma)T$, where Γ is an α -ary context and T a base kind.

2.5. Judgement Forms

There are three *primitive judgement forms* in TF:

$$\begin{aligned} & \Gamma \text{ valid} \\ & \Gamma \vdash M : T \\ & \Gamma \vdash M = N : T \end{aligned}$$

where Γ is a context, M and N are objects, and T is a base kind. These are intended to express that Γ is a valid context; that the object M has kind T under the context Γ ; and that the objects M and N are equal objects of kind T under Γ , respectively.

We now introduce *defined judgement forms* to deal with the abstractions and product kinds of higher arity:

$$\Gamma \Vdash K \text{ kind}; \quad \Gamma \Vdash K = K'; \quad \Gamma \Vdash F : K; \quad \Gamma \Vdash F = G : K \text{ .}$$

Each of these judgements is defined to be a set of primitive judgements. We shall always use the double turnstile \Vdash to indicate a defined judgement form.

For any base kind T , the defined judgement $\Gamma \Vdash T \text{ kind}$ is defined as follows:

$$\begin{aligned} (\Gamma \Vdash \mathbf{Type} \text{ kind}) &= \{\Gamma \text{ valid}\} \\ (\Gamma \Vdash \text{El}(A) \text{ kind}) &= \{\Gamma \vdash A : \mathbf{Type}\} \end{aligned}$$

For any α -ary product kind K , the judgement $\Gamma \Vdash K \text{ kind}$ is defined by:

$$(\Gamma \Vdash (\Delta)T \text{ kind}) = (\Gamma, \Delta \Vdash T \text{ kind}) \text{ .}$$

Equality of base kinds is defined by:

$$\begin{aligned} (\Gamma \Vdash \mathbf{Type} = \mathbf{Type}) &= \{\Gamma \text{ valid}\} \\ (\Gamma \Vdash \text{El}(A) = \text{El}(B)) &= \{\Gamma \vdash A = B : \mathbf{Type}\} \end{aligned}$$

We leave ' $\Gamma \Vdash \mathbf{Type} = \text{El}(B)$ ' and ' $\Gamma \Vdash \text{El}(A) = \mathbf{Type}$ ' undefined.

Equality of product kinds and contexts is defined recursively by

$$(\Gamma \Vdash (\Delta)T = (\Delta')T') = (\Gamma \Vdash \Delta = \Delta') \cup \{\Gamma \Vdash T = T'\}$$

$$\begin{aligned} (\Gamma \Vdash \langle \rangle = \langle \rangle) &= \{\Gamma \text{ valid}\} \\ (\Gamma \Vdash \Delta, x : K = \Delta', x : K') &= (\Gamma \Vdash \Delta = \Delta') \cup (\Gamma, \Delta \Vdash K = K') \end{aligned}$$

For example, the defined judgement $\Gamma \Vdash (x : A)B = (x : C)D$ is defined to be the set

$$\{\Gamma \text{ valid}, \Gamma \vdash A = C : \mathbf{Type}, \Gamma, x : A \vdash B = D : \mathbf{Type}\} \text{ .}$$

The judgement $\Gamma \Vdash (x : A)B = (x : C)\mathbf{Type}$ is undefined.

We introduce defined judgement forms $\Gamma \Vdash F : K$ and $\Gamma \Vdash F = G : K$ for the inhabitation of a product kind K by an abstraction F , and the equality of two abstractions F and G of product kind K ; here, F , G and K must all have the same arity.

$$\begin{aligned} (\Gamma \Vdash [\vec{x}]M : (\Delta)T) &= \{\Gamma, \Delta \vdash M : T\} \\ (\Gamma \Vdash [\vec{x}]M = [\vec{x}]N : (\Delta)T) &= \{\Gamma, \Delta \vdash M = N : T\} \end{aligned}$$

We assume here that we have applied α -conversion to ensure that the same variable sequence \vec{x} is used in both $[\vec{x}]P$ and $[\vec{x}]Q$, and is also the domain of the context Δ .

Finally, we introduce judgement forms

- $\Gamma \Vdash \vec{F} :: \Delta$, denoting that \vec{F} *satisfies* the context Δ ; that is, \vec{F} is a sequence of abstractions whose kinds are those given by the context Δ ;
- $\Gamma \Vdash \vec{F} = \vec{G} :: \Delta$, denoting that \vec{F} and \vec{G} are two equal abstraction sequences that satisfy Δ .

The judgement forms are defined as follows:

$$\begin{aligned} (\Gamma \Vdash \langle \rangle :: \langle \rangle) &= \{\Gamma \text{ valid}\} \\ (\Gamma \Vdash \vec{F}, F_0 :: \Delta, x : K) &= (\Gamma \Vdash \vec{F} :: \Delta) \cup (\Gamma \Vdash F_0 : \{\vec{F}/\Delta\}K) \\ \Gamma \Vdash \langle \rangle = \langle \rangle :: \langle \rangle &= \{\Gamma \text{ valid}\} \\ \Gamma \Vdash (\vec{F}, F_0) = (\vec{G}, G_0) :: (\Delta, x : K) &= (\Gamma \Vdash \vec{F} = \vec{G} :: \Delta) \cup (\Gamma \Vdash F_0 = G_0 : \{\vec{F}/\Delta\}K) \end{aligned}$$

2.6. Rules of Deduction

We are finally able to give the rules of deduction of TF. They are listed in Figure 1. They consist of the rules (emp) and (ctxt) determining when a context is valid; (var) and (var_eq), the typing and congruence rules for the application of a variable; (ref), (sym) and (trans), which ensure that the judgemental equality is an equivalence relation; and (conv) and (conv_eq), which ensure that equal kinds have the same objects.

We note in passing how few rules there are compared to logical frameworks of similar expressiveness such as LF [6] and ELF [2]. In particular, the two rules (var) and (var_eq) do all the work normally done by the rules governing typing and congruence of applications and abstractions, and β - and η -contractions. We have shifted this burden from the rules of deduction to the syntax.

2.6.1. Type Theory Specifications

An object theory is represented in TF by extending the logical framework with several new rules of deduction, representing the formation of the terms and types of the object theory and the computation rules of the object theory.

(emp_ctxt)	$\overline{\langle \rangle \text{ valid}}$	
(ctxt)	$\frac{\Gamma \Vdash K \text{ kind}}{\Gamma, x : K \text{ valid}}$	$(x \notin \text{dom } \Gamma)$
(var)	$\frac{\Gamma \Vdash \vec{F} :: \Delta}{\Gamma \vdash x\vec{F} : \{\vec{F}/\Delta\}T}$	$(x : (\Delta)T \in \Gamma)$
(var_eq)	$\frac{\Gamma \Vdash \vec{F} = \vec{G} :: \Delta}{\Gamma \vdash x\vec{F} = x\vec{G} : \{\vec{F}/\Delta\}T}$	$(x : (\Delta)T \in \Gamma)$
(ref)	$\frac{\Gamma \vdash M : T}{\Gamma \vdash M = M : T}$	
(sym)	$\frac{\Gamma \vdash M = N : T}{\Gamma \vdash N = M : T}$	
(trans)	$\frac{\Gamma \vdash M = N : T \quad \Gamma \vdash N = P : T}{\Gamma \vdash M = P : T}$	
(conv)	$\frac{\Gamma \vdash M : \text{El}(A) \quad \Gamma \vdash A = B : \mathbf{Type}}{\Gamma \vdash M : \text{El}(B)}$	
(conv_eq)	$\frac{\Gamma \vdash M = N : \text{El}(A) \quad \Gamma \vdash A = B : \mathbf{Type}}{\Gamma \vdash M = N : \text{El}(B)}$	

Figure 1: Rules of Deduction of TF

Formally, a *type theory specification* in TF is a set of *declarations*, of two possible forms:

- *constant declarations* of the form

$$c : K$$

where c is a constant and K a kind of the same arity; and

- *equation declarations* of the form

$$(\Delta)(M = N : T)$$

where Δ is a context, M and N objects and T a base kind.

The intention is that the constant declarations represent the term- and type-constructors of the object theory, and the equation declarations represent the computation rules of the object theory.

Making the constant declaration $c : (\Delta)T$ has the effect of adding the following two rules of deduction to the framework (c.f. the rules (var) and (var_eq)):

$$\begin{array}{c} \text{(const)} \frac{\Gamma \Vdash \vec{F} :: \Delta}{\Gamma \vdash c\vec{F} : \{\vec{F}/\Delta\}T} \quad \text{(const_eq)} \frac{\Gamma \Vdash \vec{F} = \vec{G} :: \Delta}{\Gamma \vdash c\vec{F} = c\vec{G} : \{\vec{F}/\Delta\}T} \end{array}$$

Making the equation declaration $(\Delta)(M = N : T)$ has the effect of adding the following rule to the framework:

$$\text{(eq)} \frac{\Gamma \Vdash \vec{F} :: \Delta}{\Gamma \vdash \{\vec{F}/\Delta\}M = \{\vec{F}/\Delta\}N : \{\vec{F}/\Delta\}T}$$

We define the *order* $o(\delta)$ of a declaration as follows: the order of $c : K$ is the order of K , and the order of $(\Delta)(M = N : T)$ is the order of Δ . The *order* $o(\mathcal{T})$ of a type theory specification \mathcal{T} is the largest n such that \mathcal{T} contains a declaration of order n , or ω if there is no such maximum.

2.7. Representing Object Theories in TF

TF is intended for representing *type theories* that have judgements of the following forms:

$$x_1 : A_1, \dots, x_n : A_n \vdash M : B \tag{3}$$

$$x_1 : A_1, \dots, x_n : A_n \vdash M = N : B \tag{4}$$

Given such a type theory T that we wish to represent in TF, we begin by forming the appropriate specification. There will be one constant declaration for each constructor in the grammar of T , and one equation declaration for each computation rule in T .

We make these declarations in such a way that:

- the objects of kind **Type** correspond to the types of T ;
- if the object $M : \mathbf{Type}$ corresponds to the type A , then the objects of kind $\text{El}(M)$ correspond to the terms of type A ;
- the judgements of T of the form (2.7) correspond to the TF judgements of the form

$$x_1 : \text{El}(A_1), \dots, x_n : \text{El}(A_n) \vdash M : \text{El}(B) \quad ; \quad (5)$$

- the judgements of T of the form (2.7) correspond to the TF judgements of the form

$$x_1 : \text{El}(A_1), \dots, x_n : \text{El}(A_n) \vdash M = N : \text{El}(B) \quad . \quad (6)$$

To specify type theories such as the Calculus of Constructions [12], ECC [6] or Martin-Löf's Type Theory without W-types [3] requires a second-order specification. To specify Martin-Löf's Type Theory with W-types requires a third-order specification. To specify UTT [6] requires a specification of order ω . These examples are described in more detail in Adams [5].

Note that the judgements of TF that represent the judgements of the object theory, those of form (5) or (6), have first-order contexts. This will be important in the following section. For many of the metatheoretic properties we investigate, we shall be able to prove that they hold for judgements with first-order contexts, but they have not yet been proved to hold for judgements with contexts of order ≥ 2 .

3. Metatheory

We can now begin to investigate the metatheoretical properties of this system. Many of these properties are more difficult to prove than the corresponding properties of a traditional logical framework; in particular, it is often the case that several properties need to be established simultaneously by a single induction. This should be seen as the 'one-time' cost of using a lambda-free framework.

3.1. Grammar

We begin by demonstrating some properties of the operations of instantiation and employment. Many of them are analogous to properties of substitution in more familiar languages; we shall point out these analogies as we proceed.

Lemma 3.1 *Let $\text{FV}(X)$ denote the set of free variables in the object or abstraction X .*

1. $\text{FV}(\{F/x\}N) \subseteq (\text{FV}(N) \setminus \{x\}) \cup \text{FV}(F)$
2. $\text{FV}(F \bullet G) \subseteq \text{FV}(F) \cup \text{FV}(G)$.

PROOF. Part 1 is proved by induction on the object N . Part 2 follows directly.

The following is the analogue of the result that, if x is not free in N , then $[M/x]N \equiv N$.

Lemma 3.2 *If x does not occur free in M , then*

$$\{F/x\}M \equiv M \text{ .}$$

PROOF. This is easily proven by induction on the object M .

Part 1 of the next lemma is the analogue of the famous Substitution Lemma.

Lemma 3.3 *Let α , β and γ be arities. Let F be an α -ary abstraction, G a β -ary abstraction, and H a $(\beta)\hat{\gamma}$ -ary abstraction. Let x be an α -ary variable and y a β -ary variable, with $x \neq y$. Let M be an object.*

1. *If x and y are distinct variables, and y does not occur free in M , then*

$$\{F/x\}\{G/y\}M \equiv \{\{F/x\}G/y\}\{F/x\}M \text{ .}$$

2. $\{F/x\}(H \bullet G) \equiv (\{F/x\}H) \bullet \{F/x\}G$.

PROOF. Both parts are proved simultaneously by induction on the sum of the orders of α and β .

Part 1 of the next lemma is the analogue of the fact that $[M/x]x \equiv M$. Part 3 is the analogue of the fact that $[x/x]M \equiv M$.

Lemma 3.4 *Let α be an arity.*

1. *For any α -ary variable x and α -ary abstraction F , $\{F/x\}x^\eta \equiv F$.*
2. *For any α -ary variable x and α -ary abstraction sequence \vec{F} , $x^\eta \bullet \vec{F} \equiv x\vec{F}$.*
3. *For any α -ary variable x and object M , $\{x^\eta/x\}M \equiv M$.*

PROOF. The three parts are proven simultaneously by induction on α . Part 3 requires a secondary induction on the object M .

3.2. Metatheoretic Properties

The following results are true in TF.

Theorem 3.5

1. (**Context Validity**) Every derivation of a judgement of the form $\Gamma, \Delta \vdash J$ has a subderivation of Γ valid.
2. Every derivation of $\Gamma, x : K, \Delta \vdash J$ has a subderivation of $\Gamma \Vdash K$ kind.
3. If $\Gamma \vdash J$ is derivable, then every free variable in the judgement body J is in the domain of Γ .
4. If $\Gamma, x : K, \Delta$ valid, then every free variable in K is in the domain of Γ .
5. (**Weakening**) If $\Gamma \vdash J$, $\Gamma \subseteq \Delta$ and Δ valid, then $\Delta \vdash J$.
6. (**Generation**) If $\Gamma \vdash x\vec{F} : T$, then there is a declaration $x : (\Delta)S$ in Γ , where

$$\Gamma \Vdash \vec{F} :: \Delta, \quad \Gamma \Vdash \{\vec{F}/\Delta\}S = T .$$

7. (**Generation**) If $\Gamma \vdash c\vec{F} : T$, then a constant declaration $c : (\Delta)S$ has been made, where

$$\Gamma \Vdash \vec{F} :: \Delta, \quad \Gamma \Vdash \{\vec{F}/\Delta\}S = T .$$

8. If $\Gamma \vdash M : T$ and $\Gamma \vdash M : T'$, then $\Gamma \Vdash T = T'$.

PROOF. The first 7 parts are each proved by a simple induction on derivations. Part 8 follows easily from parts 6 and 7.

The other metatheoretic properties of TF are very difficult to establish. We have not been able to prove the following properties in full generality, but only under a set of restrictions on the type theory specification and context.

Definition 3.6 (Good Specification). Let \mathcal{T} be a type theory specification in TF.

1. We say that \mathcal{T} is *orderable* iff there exists a well-ordering \prec on the declarations of \mathcal{T} such that:
 - (a) For every constant declaration $\delta \equiv (c : (\Delta)T)$, it is possible to derive $\Delta \Vdash T$ kind using only the declarations δ' such that $\delta' \prec \delta$.
 - (b) For every equation declaration $\delta \equiv (\Delta)(M = N : T)$, it is possible to derive $\Delta \vdash M : T$, $\Delta \vdash N : T$ and $\Delta \Vdash T$ kind using only the declarations δ' such that $\delta' \prec \delta$.
2. We say that \mathcal{T} is *n-good* iff, whenever Γ is a context of order $\leq n$ and $\Gamma \vdash M = N : T$, then $\Gamma \vdash M : T$ and $\Gamma \vdash N : T$.
3. We say that \mathcal{T} is *good* iff \mathcal{T} is *n-good* for every natural number n .

It is difficult to find general conditions under which we can prove that a specification is good. So far, we are able to do so for two large classes of specifications:

Theorem 3.7

1. If \mathcal{T} contains no equation declarations, then \mathcal{T} is good.
2. If \mathcal{T} is orderable and $o(\mathcal{T}) \leq 2$, then \mathcal{T} is 2-good.

PROOF.

1. A simple proof by induction on derivations shows that, whenever $\Gamma \vdash M = N : T$, then $M \equiv N$ and $\Gamma \vdash M : T$.
2. See Appendix B.

Theorem 3.8 *Let \mathcal{T} be a type theory specification. Suppose \mathcal{T} is n -good, and $\Gamma, x : K, \Delta$ is a context of order $\leq n$.*

1. (**Cut**) *If $\Gamma, x : K, \Delta \vdash J$ and $\Gamma \Vdash F : K$ then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}J$.*
2. (**Functionality**) *If $\Gamma, x : K, \Delta \vdash M : T$ and $\Gamma \Vdash F = G : K$ then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}M = \{G/x\}M : \{F/x\}T$.*
3. (**Context Conversion**) *If $\Gamma, x : K, \Delta \vdash J$ and $\Gamma \Vdash K = K'$ then $\Gamma, x : K', \Delta \vdash J$.*

PROOF. See Appendix A.

Once we have got past this hurdle, other properties of TF follow rapidly.

Theorem 3.9 (Type Validity) *Suppose that \mathcal{T} is an n -good specification, and*

- *for every constant declaration $c : K$ in \mathcal{T} , we have $\Vdash K$ kind;*
- *for every equation declaration $(\Delta)(M = N : T)$ in \mathcal{T} , we have $\Delta \Vdash T$ kind.*

Then, whenever $o(\Gamma) \leq n$, if $\Gamma \vdash M : \text{El}(A)$ or $\Gamma \vdash M = N : \text{El}(A)$, we have $\Gamma \vdash A : \mathbf{Type}$.

PROOF. The proof is by induction on derivations. The cases (const) and (const_eq) use the first hypothesis with Cut and Functionality respectively. The case (eq) uses the second hypothesis with Cut. The other cases are all trivial.

Theorem 3.10 (Kind Validity) *Suppose \mathcal{T} is n -good and $o(\Gamma) \leq n$. Then the following rules are admissible.*

$$\frac{\Gamma \Vdash F : K}{\Gamma \Vdash K \text{ kind}} \quad \frac{\Gamma \Vdash F = G : K}{\Gamma \Vdash K \text{ kind}}$$

PROOF. Both rules are proved admissible simultaneously by induction on the derivation of the premise. The case of the rule (var_eq) requires Equation Validity.

4. The Church-Typed TF

The version of TF we have described is *Curry-typed*; that is, the bound variables in abstractions are not annotated with their kinds. We can also construct a *Church-typed* version of TF, in which objects have the form

$$z[[x_{11} : K_{11}, \dots, x_{1r_1} : K_{1r_1}]M_1, \dots, [x_{n1} : K_{n1}, \dots, x_{nr_n} : K_{nr_n}]M_n] .$$

We shall call the Church-typed version of TF by the name TF_k ². In this section, we shall give the definition of TF_k , prove its metatheoretic properties, and define mutually inverse translations between TF and TF_k that show that the two systems are in some sense equivalent.

It is very convenient to have available two versions of a lambda-free logical framework, and to be able to switch between them at will. For example, when embedding a lambda-free framework in a traditional framework, it is easier to define translations *into* the Curry-typed version, and *from* the Church-typed version. We shall be in just this situation when we come to embed TF in LF.

4.1. Grammar

In TF_k , the sets of *objects*, *abstractions*, *abstraction sequences*, *contexts* and *kinds* are all defined simultaneously as follows.

Objects An *object* has the form $z\vec{F}$, where z is an α -ary variable or constant and \vec{F} an α -ary abstraction sequence, for some arity α .

Abstractions An α -ary *abstraction* has the form $[\Delta]M$, where Δ is an α -ary context and M an object.

Abstraction Sequences An $(\alpha_1, \dots, \alpha_n)$ -ary *abstraction sequence* has the form $\langle F_1, \dots, F_n \rangle$, where each F_i is an α_i -ary abstraction.

Contexts An $(\alpha_1, \dots, \alpha_n)$ -ary *context* has the form $x_1 : K_1, \dots, x_n : K_n$, where each x_i is an α_i -ary variable and K_i an α_i -ary kind, with the x_i s all distinct.

Kinds An α -ary *kind* has the form $(\Delta)\text{Type}$ or $(\Delta)\text{El}(M)$, where Δ is an α -ary context and M an object.

In an abstraction $[x_1 : K_1, \dots, x_n : K_n]M$ or a kind $(x_1 : K_1, \dots, x_n : K_n)T$, each variable x_i is bound wherever it occurs in K_{i+1} , K_{i+2} , \dots , K_n , and M . We identify all these expressions up to α -conversion.

The *η -long form* of a symbol in TF_k must be defined with reference to some kind. For z an α -ary variable or constant and K an α -ary kind, we define the

²The ‘k’ here stands for ‘kind’, as we include the kind labels in abstractions. This system was named TF_c in Adams [5], the ‘c’ standing for ‘Church’. I have decided to abandon this name, as ‘c’ could just as well stand for ‘Curry’!

α -ary abstraction z^K , the η -long form of z considered as being of kind K , by recursion on α as follows.

$$z^{(x_1:K_1, \dots, x_n:K_n)T} \equiv [x_1 : K_1, \dots, x_n : K_n]z[x_1^{K_1}, \dots, x_n^{K_n}] .$$

The definitions of instantiation and employment in TF_k are very similar to those in TF .

$$\{F/x\}z[G_1, \dots, G_n] \equiv z[\{F/x\}G_1, \dots, \{F/x\}G_n] \quad (z \neq x)$$

If $F \equiv [t_1 : K_1, \dots, t_n : K_n]M$,

$$\begin{aligned} \{F/x\}x[G_1, \dots, G_n] &\equiv \{\{F/x\}G_1/t_1\} \cdots \{\{F/x\}G_n/t_n\}M \\ ([x : K]F) \bullet G &\equiv \{G/x\}F \end{aligned}$$

As in TF , there are three primitive judgement forms in TF_k :

$$\Gamma \text{ valid} \quad \Gamma \vdash M : T \quad \Gamma \vdash M = N : T$$

where Γ is a context, M and N objects and T a base kind.

We define the judgement forms $\Gamma \Vdash K$ kind, $\Gamma \Vdash K = K'$ and $\Gamma \Vdash \Delta = \Delta'$ just as we did for TF .

The judgement form $\Gamma \Vdash F : K$, where F is an α -ary abstraction and K an α -ary kind, is defined as follows.

$$(\Gamma \Vdash [\Delta]M : (\Delta')T) = (\Gamma \Vdash \Delta' = \Delta) \cup \{\Gamma, \Delta' \vdash M : T\} .$$

The judgement form $\Gamma \Vdash F = G : K$, where F and G are α -ary abstractions and K an α -ary kind, is defined as follows.

$$\begin{aligned} (\Gamma \Vdash [\Delta_1]M = [\Delta_2]N : (\Delta_3)T) \\ = (\Gamma \Vdash \Delta_3 = \Delta_1) \cup (\Gamma \Vdash \Delta_3 = \Delta_2) \cup \{\Gamma, \Delta_3 \vdash M = N : T\} \end{aligned}$$

The judgement form $\Gamma \Vdash \vec{F} :: \Delta$, where \vec{F} is an α -ary abstraction sequence and Δ an α -ary context, is defined by recursion on α as follows.

$$\begin{aligned} (\Gamma \Vdash \langle \rangle :: \langle \rangle) &= \{\Gamma \text{ valid}\} \\ (\Gamma \Vdash \vec{F}, F_0 :: \Delta, x : K) &= (\Gamma \Vdash \vec{F} :: \Delta) \\ &\quad \cup (\Gamma \Vdash F_0 : \{\vec{F}/\Delta\}K) \end{aligned}$$

The judgement form $\Gamma \Vdash \vec{F} = \vec{G} :: \Delta$, where \vec{F} and \vec{G} are α -ary abstraction sequences and Δ an α -ary context, is defined by recursion on α as follows.

$$\begin{aligned} (\Gamma \Vdash \langle \rangle = \langle \rangle :: \langle \rangle) &= \{\Gamma \text{ valid}\} \\ (\Gamma \Vdash \vec{F}, F_0 = \vec{G}, G_0 :: \Delta, x : K) &= (\Gamma \Vdash \vec{F} = \vec{G} :: \Delta) \\ &\quad \cup (\Gamma \Vdash F_0 = G_0 : \{\vec{F}/\Delta\}K) \end{aligned}$$

Rules of Deduction. The rules of deduction of TF_k look exactly the same as those of TF , as given in Fig. 1. The rules (ctxt), (var) and (var_eq) of course use the new definitions of the defined judgement forms $\Gamma \Vdash K \text{ kind}$, $\Gamma \Vdash \vec{F} :: \Delta$ and $\Gamma \Vdash \vec{F} = \vec{G} :: \Delta$.

Object theories are declared in TF_k in the same way as in TF : we make a number of *constant declarations* $c : (\Delta)T$, which has the effect of introducing the rules (const) and (const_eq), and *equation declarations* $(\Delta)(M = N : T)$, which has the effect of introducing the rule (eq), as given in Section 2.6.1. Again, in TF_k these rules use the new definitions of the defined judgement forms.

Metatheory. All the properties of TF we proved in Section 3 hold in TF_k too. The proofs follow the same pattern; we have indicated in Appendix A the places where the details differ.

4.2. Translations between TF and TF_k

The systems TF and TF_k are equivalent, in the following sense. Given any derivable judgement in TF_k , erasing the kind labels on variables gives a derivable judgement in TF . Conversely, given any derivable judgement in TF , there is a way of filling in the kind labels on the variables to yield a derivable judgement in TF_k ; further, the choice of kind labels is unique up to equality in TF_k .

This fact is very convenient when working with lambda-free logical frameworks, as it allows us to switch between TF and TF_k more or less at will, effectively treating them as if they were the same system.

In this section, we shall formally establish the equivalence of TF and TF_k by defining translations between the two.

The translation from TF_k to TF consists simply of erasing the kind labels:

Definition 4.1. For every entity (object, abstraction, abstraction sequence, kind, context, or judgement) X in TF_k , let $|X|$ denote the entity obtained by erasing the kind labels on the bound variables in abstractions.

Given a type theory specification \mathcal{T} in TF_k , let $|\mathcal{T}|$ denote the type theory specification in TF formed by erasing the kind labels on the bound variables in abstractions within the declarations of \mathcal{T} .

It is straightforward to show that this translation is sound:

Theorem 4.2 *Let \mathcal{T} be a type theory specification in TF_k , and let J be a judgement that is derivable under \mathcal{T} . Then $|J|$ is a derivable judgement in TF under the type theory specification $|\mathcal{T}|$.*

PROOF. The proof consists of observing that the image of a primitive rule of deduction in TF_k under $||$ is a primitive rule of deduction in TF , and the image of any of the rules introduced by \mathcal{T} under $||$ is a rule introduced by $|\mathcal{T}|$.

Defining the translation in the other direction is harder. We shall define the translation ‘ \mathcal{L} ’ from TF to TF_k , which fills in the kind labels on the bound variables. Whenever we encounter an object of the form $x[\cdots, [y_1, \dots, y_n]M, \cdots]$, we discover the kinds of y_1, \dots, y_n by looking up the kind of x in the current context. Similarly, we handle objects of the form $c[\cdots]$ by looking up the kind of c in the specification.

Let us say that an object, abstraction or abstraction sequence X in TF is *defined* relative to the specification \mathcal{T} and context Γ if and only if every constant that occurs in X is declared in \mathcal{T} , and every free variable in X is declared in Γ . Let us also say that a context $\Delta \equiv x_1 : K_1, \dots, x_n : K_n$ is *defined* relative to Γ and \mathcal{T} if and only if, for each i , K_i is defined relative to the context $\Gamma, x_1 : K_1, \dots, x_{i-1} : K_{i-1}$ and \mathcal{T} . Let us say that a judgement $\Gamma \vdash J$ is *defined* relative to \mathcal{T} if and only if Γ is defined relative to \mathcal{T} , every constant that occurs in J is declared in \mathcal{T} , and every free variable in J is declared in Γ .

Let us say that the specification \mathcal{T} is *consistent* if and only if:

- for each constant declaration $c : K$, the kind K is defined relative to the empty context and \mathcal{T} ;
- for each equation declaration $(\Delta)(M = N : T)$, the context Δ is defined relative to \mathcal{T} , and M , N and T are defined relative to Δ and \mathcal{T} .

Now, given a consistent specification \mathcal{T} in TF, we shall define the following.

- For every context Γ defined relative to \mathcal{T} , and every object M defined relative to \mathcal{T} and Γ , an object $\mathcal{L}_\Gamma(M)$ in TF_k .
- For every abstraction F and kind K of the same arity defined relative to Γ and \mathcal{T} , an abstraction $\mathcal{L}_\Gamma^K(F)$ in TF_k . We think of K as the intended kind of F .
- For every abstraction sequence \vec{F} and context Δ of the same arity defined relative to Γ and \mathcal{T} , an abstraction sequence $\mathcal{L}_\Gamma^\Delta(\vec{F})$ in TF_k . We think of Δ as giving the intended kinds of the abstractions \vec{F} .
- For every kind K defined relative to Γ and \mathcal{T} , a kind $\mathcal{L}_\Gamma(K)$ in TF_k .
- For every context Γ defined relative to \mathcal{T} , a context $\mathcal{L}(\Gamma)$ in TF_k .
- For every judgement J defined relative to \mathcal{T} , a judgement $\mathcal{L}(J)$ in TF_k .

The definition is as follows.

$$\begin{aligned}
\mathcal{L}_\Gamma(c\vec{F}) &\equiv c[\mathcal{L}_\Gamma^\Delta(\vec{F})] && (c : (\Delta)T \text{ declared in } \mathcal{T}) \\
\mathcal{L}_\Gamma(x\vec{F}) &\equiv x[\mathcal{L}_\Gamma^\Delta(\vec{F})] && (x : (\Delta)T \text{ declared in } \Gamma) \\
\\
\mathcal{L}_\Gamma^T(M) &\equiv \mathcal{L}_\Gamma(M) \\
\mathcal{L}_\Gamma^{(x:K)K'}([x]F) &\equiv [x : \mathcal{L}_\Gamma(K)]\mathcal{L}_{\Gamma,x:K}^{K'}(F) \\
\\
\mathcal{L}_\Gamma^\langle \rangle(\langle \rangle) &\equiv \langle \rangle \\
\mathcal{L}_\Gamma^{\Delta,x:K}(\vec{F}, G) &\equiv \mathcal{L}_\Gamma^\Delta(\vec{F}), \mathcal{L}_\Gamma^{\{\vec{F}/\Delta\}K}(G) \\
\\
\mathcal{L}_\Gamma(\mathbf{Type}) &\equiv \mathbf{Type} \\
\mathcal{L}_\Gamma(\text{El}(M)) &\equiv \text{El}(\mathcal{L}_\Gamma(M)) \\
\mathcal{L}_\Gamma((x : K)K') &\equiv (x : \mathcal{L}_\Gamma(K))\mathcal{L}_{\Gamma,x:K}(K') \\
\\
\mathcal{L}(\langle \rangle) &\equiv \langle \rangle \\
\mathcal{L}(\Gamma, x : K) &\equiv \mathcal{L}(\Gamma), \mathcal{L}_\Gamma(K) \\
\\
\mathcal{L}(\Gamma \text{ valid}) &\equiv \mathcal{L}(\Gamma) \text{ valid} \\
\mathcal{L}(\Gamma \vdash M : T) &\equiv \mathcal{L}(\Gamma) \vdash \mathcal{L}_\Gamma(M) : \mathcal{L}_\Gamma(T) \\
\mathcal{L}(\Gamma \vdash M = N : T) &\equiv \mathcal{L}(\Gamma) \vdash \mathcal{L}_\Gamma(M) = \mathcal{L}_\Gamma(N) : \mathcal{L}_\Gamma(T)
\end{aligned}$$

Given a consistent specification \mathcal{S} in TF, let $\mathcal{L}(\mathcal{S})$ be the following type theory specification in TF_k .

- For every constant declaration $c : K$ in \mathcal{S} , declare $c : \mathcal{L}_\langle \rangle(K)$.
- For every equation declaration $(\Delta)(M = N : T)$ in \mathcal{S} , declare $(\mathcal{L}(\Delta))(\mathcal{L}_\Delta(M) = \mathcal{L}_\Delta(N) : \mathcal{L}_\Delta(T))$.

We can show that this translation is sound after proving a number of lemmas.

Lemma 4.3 *If M is defined relative to both Γ and Δ , and Γ and Δ agree on every free variable in M , then $\mathcal{L}_\Gamma(M) \equiv \mathcal{L}_\Delta(M)$. In particular, if M is defined relative to Γ and $\Gamma \subseteq \Delta$, then $\mathcal{L}_\Gamma(M) \equiv \mathcal{L}_\Delta(M)$.*

PROOF. An easy induction on M .

Lemma 4.4 *For each of the following equations, if the left-hand side is defined then so is the right-hand side, in which case the two are equal.*

$$\begin{aligned}\{\mathcal{L}_\Gamma^K(F)/x\}\mathcal{L}_{\Gamma,x:K,\Delta}(X) &\equiv \mathcal{L}_{\Gamma,\{F/x\}\Delta}(\{F/x\}X) \\ \{\mathcal{L}_\Gamma^K(F)/x\}\mathcal{L}_{\Gamma,x:K,\Delta}^{K'}(G) &\equiv \mathcal{L}_{\Gamma,\{F/x\}\Delta}^{\{F/x\}K'}(\{F/x\}G) \\ \{\mathcal{L}_\Gamma^K(F)/x\}\mathcal{L}_{\Gamma,x:K,\Delta}^\Theta(\vec{G}) &\equiv \mathcal{L}_{\Gamma,\{F/x\}\Delta}^{\{F/x\}\Theta}(\{F/x\}\vec{G})\end{aligned}$$

where X is an object, kind or context.

PROOF. The five equations are proved simultaneously by a double induction on the arity of K , then the size of X , G or \vec{G} . We give the calculation for one case, the case where X is an object of the form $x\vec{G}$. Let $K \equiv (\Theta)T$ and $F \equiv [\text{dom } \Theta]N$.

$$\begin{aligned}\{\mathcal{L}(F)/x\}\mathcal{L}(x\vec{G}) &\equiv \mathcal{L}(F) \bullet \{\mathcal{L}(F)/x\}\mathcal{L}(\vec{G}) \\ &\equiv \mathcal{L}(F) \bullet \mathcal{L}(\{F/x\}\vec{G}) \quad (\text{i.h. on } X) \\ &\equiv \{\mathcal{L}(\{F/x\}\vec{G})/\Theta\}\mathcal{L}(N) \\ &\equiv \mathcal{L}(\{\{F/x\}\vec{G}/\Theta\}N) \quad (\text{i.h. on arity}) \\ &\equiv \mathcal{L}(\{F/x\}x\vec{G})\end{aligned}$$

The following lemma shows how we can change the subscript and superscript on an abstraction $\mathcal{L}_\Gamma^K(F)$. Roughly, it can be read as: if $\mathcal{L}(\Gamma) = \mathcal{L}(\Gamma')$ and $\mathcal{L}(K) = \mathcal{L}(K')$, then $\mathcal{L}_\Gamma^K(F) = \mathcal{L}_{\Gamma'}^{K'}(F)$.

Lemma 4.5 *The following rule of deduction is admissible in TF_k .*

$$\frac{\begin{array}{c} \mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma^K(F) : \mathcal{L}_\Gamma(K) \\ \Vdash \mathcal{L}_\Diamond(\Gamma) = \mathcal{L}_\Diamond(\Gamma') \quad \mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma(K) = \mathcal{L}_{\Gamma'}(K') \end{array}}{\mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma^K(F) = \mathcal{L}_{\Gamma'}^{K'}(F) : \mathcal{L}_\Gamma(K)}$$

PROOF. We prove that this rule and the following two are admissible.

$$\frac{\begin{array}{c} \mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma(M) : \mathcal{L}_\Gamma(T) \\ \Vdash \mathcal{L}_\Diamond(\Gamma) = \mathcal{L}_\Diamond(\Gamma') \end{array}}{\mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma(M) = \mathcal{L}_{\Gamma'}(M) : \mathcal{L}_\Gamma(T)} \quad (7)$$

$$\frac{\begin{array}{c} \mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma^\Theta(\vec{F}) :: \mathcal{L}_\Gamma(\Theta) \\ \Vdash \mathcal{L}_\Diamond(\Gamma, \Theta) = \mathcal{L}_\Diamond(\Gamma', \Theta') \end{array}}{\mathcal{L}_\Diamond(\Gamma) \Vdash \mathcal{L}_\Gamma^\Theta(\vec{F}) = \mathcal{L}_{\Gamma'}^{\Theta'}(\vec{F}) :: \mathcal{L}_\Gamma(\Theta)}$$

The three rules are proved admissible simultaneously by induction on the size of $\mathcal{L}_\Gamma^K(F)$, $\mathcal{L}_\Gamma(M)$ and $\mathcal{L}_\Gamma^\Theta(\vec{F})$. We give here the details for the case for (7) where M has the form $x\vec{F}$.

Let x have kind $(\Theta)S$ in Γ and $(\Theta')S'$ in Γ' . We are given that $\mathcal{L}(\Gamma) \vdash \mathcal{L}_\Gamma(M) : \mathcal{L}(T)$. Therefore, by Generation,

$$\mathcal{L}(\Gamma) \Vdash \mathcal{L}_\Gamma^\Theta(\vec{F}) :: \mathcal{L}(\Theta) \quad \mathcal{L}(\Gamma) \Vdash \{\mathcal{L}_\Gamma^\Theta(\vec{F})/\Theta\}\mathcal{L}(S) = \mathcal{L}(T) .$$

The induction hypothesis gives

$$\mathcal{L}(\Gamma) \Vdash \mathcal{L}_\Gamma^\Theta(\vec{F}) = \mathcal{L}_{\Gamma'}^{\Theta'}(\vec{F}) :: \mathcal{L}(\Theta) .$$

and the desired result follows by (var.eq) and (conv.eq).

Lemma 4.6 *Let T be an n -good declaration in TF_k . The following rules of deduction are admissible in TF_k .*

$$\begin{aligned} (\mathcal{L}\text{-seq}) \quad & \frac{\mathcal{L}(\Gamma \Vdash \vec{F} :: \Theta)}{\mathcal{L}_{\langle \rangle}(\Gamma) \Vdash \mathcal{L}_\Gamma^\Theta(\vec{F}) :: \mathcal{L}_\Gamma(\Theta)} \\ (\mathcal{L}\text{-sepeq}) \quad & \frac{\mathcal{L}(\Gamma \Vdash \vec{F} = \vec{G} :: \Theta)}{\mathcal{L}_{\langle \rangle}(\Gamma) \Vdash \mathcal{L}_\Gamma^\Theta(\vec{F}) = \mathcal{L}_\Gamma^\Theta(\vec{G}) :: \mathcal{L}_\Gamma(\Theta)} \end{aligned}$$

where Γ , \vec{F} , \vec{G} and Θ are of order $\leq n$.

PROOF. We first prove the following two rules are admissible.

$$\begin{aligned} (\mathcal{L}\text{-abs}) \quad & \frac{\mathcal{L}(\Gamma \Vdash F : K)}{\mathcal{L}_{\langle \rangle}(\Gamma) \Vdash \mathcal{L}_\Gamma^K(F) : \mathcal{L}_\Gamma(K)} \\ (\mathcal{L}\text{-abseq}) \quad & \frac{\mathcal{L}(\Gamma \Vdash F = G : K)}{\mathcal{L}_{\langle \rangle}(\Gamma) \Vdash \mathcal{L}_\Gamma^K(F) = \mathcal{L}_\Gamma^K(G) : \mathcal{L}_\Gamma(K)} \end{aligned}$$

For the first of these rules, if $K \equiv (\Theta)T$ and $F \equiv [\text{dom } \Theta]M$, then the premise is $\mathcal{L}(\Gamma), \mathcal{L}(\Theta) \vdash \mathcal{L}(M) : \mathcal{L}(T)$, and the conclusion is

$$(\mathcal{L}(\Gamma) \Vdash \mathcal{L}(\Theta) = \mathcal{L}(\Theta)) \cup \{\mathcal{L}(\Gamma), \mathcal{L}(\Theta) \vdash \mathcal{L}(M) : \mathcal{L}(T)\}$$

which follows using Context Validity and (ref). The proof for the second rule is similar.

The rules $(\mathcal{L}\text{-seq})$ and $(\mathcal{L}\text{-sepeq})$ are each proved admissible by induction on the length of \vec{F} . We give the details for the rule $(\mathcal{L}\text{-sepeq})$ where the length of \vec{F} is greater than 0. Suppose now that $\vec{F} \equiv \vec{F}_0, F_1$; $\vec{G} \equiv \vec{G}_0, G_1$; and $\Theta \equiv \Theta_0, x : K_1$. The premises are

$$\mathcal{L}(\Gamma \Vdash \vec{F}_0 = \vec{G}_0 :: \Theta_0) \cup \mathcal{L}(\Gamma \Vdash F_1 = G_1 : \{\vec{F}_0/\Theta_0\}K_1)$$

and the conclusion is

$$\begin{aligned} & (\mathcal{L}(\Gamma) \Vdash \mathcal{L}(\vec{F}_0) = \mathcal{L}(\vec{G}_0) :: \mathcal{L}(\Theta_0)) \\ \cup \quad & (\mathcal{L}(\Gamma) \Vdash \mathcal{L}_\Gamma^{\{\vec{F}_0/\Theta_0\}K_1}(F_1) = \mathcal{L}_\Gamma^{\{\vec{G}_0/\Theta_0\}K_1}(G_1) : \mathcal{L}_\Gamma(\{\vec{F}_0/\Theta_0\}K_1)) . \end{aligned}$$

This follows, using the induction hypothesis, the rule ($\mathcal{L}_{\text{abseq}}$) and Lemma 4.5, once we have shown

$$\mathcal{L}(\Gamma) \Vdash \mathcal{L}(\{\vec{F}_0/\Theta_0\}K_1) = \mathcal{L}(\{\vec{G}_0/\Theta_0\}K_1) \ .$$

By Lemma 4.4, this is

$$\mathcal{L}(\Gamma) \Vdash \{\mathcal{L}(\vec{F}_0)/\Theta_0\}\mathcal{L}(K_1) = \{\mathcal{L}(\vec{G}_0)/\Theta_0\}\mathcal{L}(K_1)$$

which is obtainable using Functionality.

Theorem 4.7 *Let \mathcal{S} be an orderable n -good type theory specification in TF in which every declaration has order $\leq n$. Assume we have declared \mathcal{S} in TF and $\mathcal{L}(\mathcal{S})$ in TF_k . Then, for every judgement J derivable in TF with context of order $\leq n$, the judgement $\mathcal{L}(J)$ is derivable in TF_k .*

PROOF. Let \prec be the given order on \mathcal{S} . For each declaration δ in \mathcal{S} , let \mathcal{S}_δ be the set of declarations δ' such that $\delta' \prec \delta$. We prove the following simultaneously by \prec -induction on δ :

1. $\mathcal{L}(\mathcal{S}_\delta)$ is an orderable n -good specification in TF_k .
2. If J is derivable in TF under \mathcal{S}_δ , and J has context of order $\leq n$, then $\mathcal{L}(J)$ is derivable under $\mathcal{L}(\mathcal{S}_\delta)$ in TF_k .

The proof of 2 is by a straightforward induction on the derivation of J . The cases (var), (const), (eq) all make use of the first rule in Lemma 4.6; the cases (var_eq) and (const_eq) make use of the second rule in that lemma.

Thus, our translations between TF and TF_k are sound. It is also easy to show that the mapping $||$ is an exact left inverse to \mathcal{L} :

Theorem 4.8

$$|\mathcal{L}_\Gamma(X)| \equiv X \quad |\mathcal{L}_\Gamma^K(F)| \equiv F \quad |\mathcal{L}_\Gamma^\Delta(\vec{F})| \equiv \vec{F}$$

where X is an object, kind or context.

PROOF. An easy induction on X , F and \vec{F} .

The mapping \mathcal{L} is *not* a left inverse to $||$ up to syntactic identity. For example,

$$\mathcal{L}_{A:\text{Type}, B:\text{Type}, C:\text{Type}}^{(x:\text{El}(A))\text{El}(C)}(|[x:\text{El}(B)]x|) \equiv [x:\text{El}(A)]x \ .$$

However, on the well-typed objects, abstractions and kinds, \mathcal{L} is a left inverse to $||$ up to *equality* in TF_k , in the following sense.

Theorem 4.9 *Let T be an n -good declaration in TF_k , and Γ, F, K, Δ have order $\leq n$.*

1. *If $\Gamma \vdash M : T$ then $\Gamma \vdash M = \mathcal{L}_{|\Gamma|}(|M|) : T$.*
2. *If $\Gamma \Vdash F : K$ then $\Gamma \Vdash F = \mathcal{L}_{|\Gamma|}^{|K|}(|F|) : K$.*
3. *If $\Gamma \Vdash \vec{F} :: \Delta$ then $\Gamma \Vdash \vec{F} = \mathcal{L}_{|\Gamma|}^{|\Delta|}(|\vec{F}|) :: \Delta$.*
4. *If $\Gamma \Vdash K$ kind then $\Gamma \Vdash K = \mathcal{L}_{|\Gamma|}(|K|)$.*
5. *If Γ, Δ valid then $\Gamma \Vdash \Delta = \mathcal{L}_{|\Gamma|}(|\Delta|)$.*

PROOF. Let $l(X)$ denote the length of an expression X . The five parts are proven simultaneously by induction on $l(\Gamma) + l(M)$, $l(\Gamma) + l(F)$, $l(\Gamma) + l(\vec{F})$, $l(\Gamma) + l(K)$, and $l(\Gamma) + l(\Delta)$. We give here the details of the first two parts.

1. Suppose $M \equiv x\vec{F}$, where $x : (\Theta)S \in \Gamma$. By Generation,

$$\Gamma \Vdash \vec{F} :: \Theta, \quad \Gamma \Vdash \{\vec{F}/\Theta\}S = T.$$

Therefore,

$$\begin{aligned} \Gamma \Vdash \vec{F} &= \mathcal{L}_{|\Gamma|}^{|\Theta|}(|\vec{F}|) :: \Theta && (\text{i.h.}) \\ \therefore \Gamma \vdash x\vec{F} &= x \left[\mathcal{L}_{|\Gamma|}^{|\Theta|}(|\vec{F}|) \right] : \{\vec{F}/\Theta\}S && (\text{var_eq}) \\ \therefore \Gamma \vdash x\vec{F} &= x \left[\mathcal{L}_{|\Gamma|}^{|\Theta|}(|\vec{F}|) \right] : T && (\text{conv_eq}) \end{aligned}$$

The case $M \equiv c\vec{F}$ is similar.

2. Let $K \equiv (\Theta)T$ and $F \equiv [\Theta']M$. We are given that

$$\Gamma \Vdash \Theta = \Theta', \quad \Gamma, \Theta \vdash M : T.$$

We must show that $\Gamma \Vdash [\Theta']M = [\mathcal{L}_{|\Gamma|}(|\Theta|)]\mathcal{L}_{|\Gamma|, |\Theta|}(|M|) : (\Theta)T$. The induction hypothesis gives us that $\Gamma, \Theta \vdash M = \mathcal{L}_{|\Gamma|, |\Theta|}(|M|) : T$; it remains to show

$$\Gamma \Vdash \Theta = \Theta'.$$

The induction hypothesis gives us that $\Gamma \Vdash \Theta' = \mathcal{L}_{|\Gamma|}(|\Theta'|)$; and $\Gamma \Vdash \Theta = \mathcal{L}_{|\Gamma|}(|\Theta|)$; it is thus sufficient to show

$$\Gamma \Vdash \mathcal{L}_{|\Gamma|}(|\Theta|) = \mathcal{L}_{|\Gamma|}(|\Theta'|).$$

Well,

$$\begin{aligned} |\Gamma| &\Vdash_{\text{TF}} |\Theta| = |\Theta'| && (\text{Theorem 4.2}) \\ \therefore \mathcal{L}_{|\Gamma|}(|\Gamma|) &\Vdash_{\text{TF}} \mathcal{L}_{|\Gamma|}(|\Theta|) = \mathcal{L}_{|\Gamma|}(|\Theta'|) && (\text{Theorem 4.7}) \\ &\Vdash_{\text{TF}} \Gamma = \mathcal{L}_{|\Gamma|}(|\Gamma|) && (\text{i.h.}) \end{aligned}$$

and the result follows by Context Conversion.

Parts 3–5 are proven similarly.

We have thus established sound translations $||$ and \mathcal{L} between TF and TF_k which are inverses of one another up to the appropriate notion of equality.

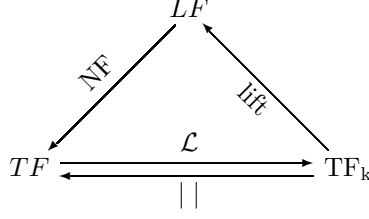


Figure 2: Translations between Logical Frameworks

5. Embedding TF in LF

Lambda-free frameworks can often be embedded within existing traditional logical frameworks; that is, given a traditional logical framework F , we can often construct a lambda-free framework (its *core*) that is, in some sense, isomorphic to a subsystem of F . More precisely, we can construct a lambda-free framework L and define translations

$$\text{NF} : F \rightarrow L, \quad \text{lift} : L \rightarrow F .$$

These translations are sound, and NF is a left inverse to ‘lift’ up to identity (α -conversion). That is, we have the following properties:

1. For every derivable judgement J in L , $\text{lift}(J)$ is derivable in F .
2. For every derivable judgement J in F , $\text{NF}(J)$ is derivable in L .
3. For every typable expression X in L , $\text{NF}(\text{lift}(X)) \equiv X$.

In many cases (particularly when F allows η -conversion) we have in addition that NF is a right inverse to lift up to the equality judgements of F :

4. For every typable expression X in F , the equality $\text{lift}(\text{NF}(X)) = X$ is derivable in F .

We can think of F as picking out, from each equivalence class of the expressions of F modulo $\beta\eta$ -convertibility, a unique representative: the β -normal, η -long form.

Establishing the above properties of the translations is not easy; it usually involves proving fairly strong properties of L and F . However, once this one-time cost has been paid, we can then use the translations to prove various properties of F more easily. It is often the case that it is easier to establish a given metatheoretic property for L than for F . Once it has been proven to hold in L , the result can then be ‘lifted’ to F ; that is, we can derive the corresponding result for F using the properties of the translations.

In this section, we shall show how TF can be embedded in this fashion within the framework LF introduced in [6], a Church-typed version of Martin-Löf’s logical framework. It will prove to be very advantageous that we have two different versions of TF; we shall define translations from TF_k to LF, and from LF to TF, as shown in Figure 2.

5.1. The Framework LF

The framework LF [6] is a Church-typed version of Martin-Löf's logical framework³. LF deals with *objects* and *kinds*, given by the following grammar:

$$\begin{aligned}\text{Kind } K &::= \mathbf{Type} \mid \text{El}(k) \mid (x : K)K \\ \text{Object } k &::= x \mid c \mid [x : K]k \mid kk\end{aligned}$$

where x is a variable and c a constant. There are five judgement forms in LF:

- Γ valid, which denotes that Γ is a valid context;
- $\Gamma \vdash K$ kind, which denotes that K is a kind under Γ ;
- $\Gamma \vdash k : K$, which denotes that k is an object of kind K under Γ ;
- $\Gamma \vdash k = k' : K$, which denotes that k and k' are equal objects of kind K under Γ ;
- $\Gamma \vdash K = K'$, which denotes that K and K' are equal kinds under Γ .

A type theory is specified in LF by giving a set of *constant declarations* $c : K$, and a set of *computation rules*

$$k = k' : K \text{ for } k_1 : K_1, \dots, k_n : K_n.$$

We shall make use of the following abbreviations when working with LF. Let Δ be the context $x_1 : K_1, \dots, x_n : K_n$, and Δ' the context $x_1 : K'_1, \dots, x_n : K'_n$. We shall write $\Gamma \Vdash \Delta = \Delta'$ for the n judgements

$$\begin{array}{ll}\Gamma & \vdash K_1 = K'_1, \\ \Gamma, x_1 : K_1 & \vdash K_2 = K'_2, \\ & \vdots \\ \Gamma, x_1 : K_1, \dots, x_{n-1} : K_{n-1} & \vdash K_n = K'_n\end{array}$$

and we shall write $\Gamma \Vdash (k_1, \dots, k_n) :: \Delta$ for the n judgements

$$\Gamma \vdash k_1 : K_1, \quad \Gamma \vdash k_2 : [k_1/x_1]K_2, \quad \dots, \quad \Gamma \vdash k_n : [k_1/x_1, \dots, k_{n-1}/x_{n-1}]K_n.$$

For the rules of deduction of LF, and how LF may be used to specify various object theories, we refer to Luo [6].

We note that, as with TF, the judgements of the object theory are represented by the LF-judgements of the form

$$\begin{aligned}x_1 : \text{El}(A_1), \dots, x_n : \text{El}(A_n) & \vdash k : \text{El}(B) \\ x_1 : \text{El}(A_1), \dots, x_n : \text{El}(A_n) & \vdash k = k' : \text{El}(B)\end{aligned}$$

and these are judgements with first-order contexts.

We shall make use of the fact that LF satisfies *Subject Reduction*:

$$\text{If } \Gamma \vdash k : K \text{ and } k \rightarrow_{\beta\eta} k', \text{ then } \Gamma \vdash k = k' : K.$$

³The framework here called LF should not be confused with the Edinburgh Logical Framework [2], which is also often referred to as LF.

5.2. Translation from TF_k to LF

We shall now define our translations between LF and the two versions of TF. The mapping from TF_k to LF, which we shall call ‘lift’, is almost trivial. We map objects and abstractions to objects, kinds to kinds, contexts to contexts and judgements to judgements as follows.

$$\begin{aligned}
\text{lift}(x[F_1, \dots, F_n]) &\equiv x\text{lift}(F_1) \cdots \text{lift}(F_n) \\
\text{lift}([\Delta]M) &\equiv [\text{lift}(\Delta)]\text{lift}(M) \\
\\
\text{lift}(\mathbf{Type}) &\equiv \mathbf{Type} \\
\text{lift}(\text{El}(M)) &\equiv \text{El}(\text{lift}(M)) \\
\text{lift}((x : K)K') &\equiv (x : \text{lift}(K))\text{lift}(K') \\
\\
\text{lift}(x_1 : K_1, \dots, x_n : K_n) &\equiv x_1 : \text{lift}(K_1), \dots, x_n : \text{lift}(K_n) \\
\\
\text{lift}(\Gamma \text{ valid}) &\equiv \text{lift}(\Gamma) \text{ valid} \\
\text{lift}(\Gamma \vdash M : T) &\equiv \text{lift}(\Gamma) \vdash \text{lift}(M) : \text{lift}(T) \\
\text{lift}(\Gamma \vdash M = N : T) &\equiv \text{lift}(\Gamma) \vdash \text{lift}(M) = \text{lift}(N) : \text{lift}(T)
\end{aligned}$$

It is relatively straightforward to establish that this translation is sound.

Lemma 5.1

$$[\text{lift}(F)/x]\text{lift}(N) \rightarrow_{\beta} \text{lift}(\{F/x\}N)$$

PROOF. The proof is by a double induction on the arity of F and x , then on the object N . We give here the details for the case $N \equiv x\vec{G}$. Let $F \equiv [\Delta]P$.

$$\begin{aligned}
[\text{lift}(F)/x]x\text{lift}(\vec{G}) &\equiv \text{lift}(F)[\text{lift}(F)/x]\text{lift}(\vec{G}) \\
&\equiv ([\text{lift}(\Delta)]\text{lift}(P))[\text{lift}(F)/x]\text{lift}(\vec{G}) \\
&\rightarrow [[\text{lift}(F)/x]\text{lift}(\vec{G})/\Delta]\text{lift}(P) \\
&\rightarrow [\text{lift}(\{F/x\}\vec{G})/\Delta]\text{lift}(P) && \text{(i.h.)} \\
&\rightarrow \text{lift}(\{\{F/x\}\vec{G}/\Delta\}P) && \text{(i.h.)} \\
&\equiv \text{lift}(\{F/x\}N)
\end{aligned}$$

Theorem 5.2 *Suppose we have declared a type theory T in TF_k , and the corresponding theory $\text{lift}(T)$ in LF . If \mathcal{J} is a derivable judgement in TF_k , then $\text{lift}(\mathcal{J})$ is derivable in LF .*

PROOF. We first prove that the following rules of deduction are admissible in LF :

$$\begin{aligned}
& (\text{lift_abs}) \frac{\text{lift}(\Gamma \Vdash F : K)}{\text{lift}(\Gamma) \vdash \text{lift}(F) : \text{lift}(K)} \\
& (\text{lift_abseq}) \frac{\text{lift}(\Gamma \Vdash F = G : K)}{\text{lift}(\Gamma) \vdash \text{lift}(F) = \text{lift}(G) : \text{lift}(K)} \\
& (\text{lift_seq}) \frac{\text{lift}(\Gamma \Vdash \vec{F} :: \Delta) \quad \text{lift}(\Gamma, \Delta \text{ valid})}{\text{lift}(\Gamma) \vdash \text{lift}(\vec{F}) :: \text{lift}(\Delta)} \\
& (\text{lift_sepeq}) \frac{\text{lift}(\Gamma \Vdash \vec{F} = \vec{G} :: \Delta) \quad \text{lift}(\Gamma, \Delta \text{ valid})}{\text{lift}(\Gamma) \vdash \text{lift}(\vec{F}) = \text{lift}(\vec{G}) :: \text{lift}(\Delta)}
\end{aligned}$$

The proof for (lift_seq) is by induction on the length of \vec{F} .

If the length is 0, both hypothesis and conclusion are that $\text{lift}(\Gamma)$ is valid.

Suppose \vec{F} is of length $n + 1$, and the result holds for abstraction sequences of length n . Let $\vec{F} \equiv \vec{F}_0, F_1$; and $\Delta \equiv \Delta_0, x : K_1$. We are given that $\text{lift}(\Gamma \Vdash \vec{F}_0 :: \Delta_0)$ is derivable, hence so is $\text{lift}(\Gamma) \vdash \text{lift}(\vec{F}_0) :: \text{lift}(\Delta_0)$ by the induction hypothesis. We also have

$$\text{lift}(\Gamma) \vdash \text{lift}(F_1) : \text{lift}(\{\vec{F}_0/\Delta_0\}K_1)$$

by part 1 and

$$\text{lift}(\Gamma), \text{lift}(\Delta_0) \vdash \text{lift}(K_1) \text{ kind}$$

by Kind Validity in LF . This yields

$$\begin{aligned}
& \text{lift}(\Gamma) \vdash [\text{lift}(\vec{F}_0)/\Delta_0]\text{lift}(K_1) \text{ kind} && (\text{substitution}) \\
\therefore \text{lift}(\Gamma) \vdash [\text{lift}(\vec{F}_0)/\Delta_0]\text{lift}(K_1) &= \text{lift}(\{\vec{F}_0/\Delta_0\}K_1) && \\
& (\text{Subject Reduction, Lemma 5.1}) && \\
\therefore \text{lift}(\Gamma) \vdash \text{lift}(F_1) : [\text{lift}(\vec{F}_0)/\Delta_0]\text{lift}(K_1) && (\text{conv})
\end{aligned}$$

as required.

The proof for (lift_sepeq) is similar, and the proofs for (lift_abs) and (lift_seq) are simple. The theorem now follows by induction on the derivation of \mathcal{J} .

5.3. Translation from LF to TF

The translation from LF to TF is more difficult to construct. It consists of reducing every entity of LF to its β -normal, η -long form.

We must first assign arities to the entities of LF, to guide us during η -expansion. We assign an arity to every kind of LF as follows:

$$\begin{aligned} \text{Ar}(\mathbf{Type}) &\equiv \mathbf{0} \\ \text{Ar}(\text{El}(k)) &\equiv \mathbf{0} \\ \text{Ar}((x : K_1)K_2) &\equiv (\text{Ar}(K_1))^\wedge \text{Ar}(K_2) \end{aligned}$$

We now define an arity $\text{Ar}_\Gamma(k)$ to some LF-contexts Γ and LF-objects k as follows:

- If $x : K$ is an entry in Γ , then $\text{Ar}_\Gamma(x) \equiv \text{Ar}(K)$.
- If c has been declared with arity K , then $\text{Ar}_\Gamma(c) \equiv \text{Ar}(K)$.
- If $\text{Ar}_{\Gamma, x:K}(k)$ is defined, then $\text{Ar}_\Gamma([x : K]k) \equiv (\text{Ar}(K))^\wedge \text{Ar}_{\Gamma, x:K}(k)$.
- If $\text{Ar}_\Gamma(k)$ and $\text{Ar}_\Gamma(k')$ is defined, and $\text{Ar}_\Gamma(k)$ has the form

$$\text{Ar}_\Gamma(k) \equiv (\text{Ar}_\Gamma(k'))^\wedge \beta$$

then $\text{Ar}_\Gamma(kk') \equiv \beta$.

We shall say that an object k is *well-aritied* if $\text{Ar}_\Gamma(k)$ is defined. We shall only be able to map well-aritied objects into TF. We can prove immediately that every object typable in LF is well-aritied.

Proposition 5.3 *In LF,*

1. *if $\Gamma \vdash k : K$ then $\text{Ar}_\Gamma(k) \equiv \text{Ar}(K)$;*
2. *if $\Gamma \vdash k = k' : K$ then $\text{Ar}_\Gamma(k) \equiv \text{Ar}_\Gamma(k') \equiv \text{Ar}(K)$;*
3. *if $\Gamma \vdash K = K'$ then $\text{Ar}(K) \equiv \text{Ar}(K')$.*

PROOF. The three statements are proven simultaneously by induction on the derivation of the premise. We need to make use of the following two auxiliary facts, which are easy to prove:

1. $\text{Ar}([k/x]K) \equiv \text{Ar}(K)$
2. If $\text{Ar}_\Gamma(k) \equiv \text{Ar}(K)$ and $\text{Ar}_{\Gamma, x:K}(k')$ is defined, then we have $\text{Ar}_\Gamma([k/x]k') \equiv \text{Ar}_{\Gamma, x:K}(k')$.

Given an object k such that $\text{Ar}_\Gamma(k) \equiv \alpha$, we define the α -ary abstraction $\text{NF}_\Gamma(k)$ in TF as follows.

$$\begin{aligned} \text{NF}_\Gamma(x) &\equiv x^\eta \\ \text{NF}_\Gamma(c) &\equiv c^\eta \\ \text{NF}_\Gamma([x : K]k) &\equiv [x]\text{NF}_{\Gamma, x:K}(k) \\ \text{NF}_\Gamma(kk') &\equiv \text{NF}_\Gamma(k) \bullet \text{NF}_\Gamma(k') \end{aligned}$$

where, in the first two clauses, x has arity $\text{Ar}_\Gamma(x)$ and c has arity $\text{Ar}_\Gamma(c)$. In the third clause, x has arity $\text{Ar}_\Gamma(K)$.

We extend the mapping NF to kinds, contexts and judgements as follows.

$$\begin{aligned}
\text{NF}_\Gamma(\mathbf{Type}) &\equiv \mathbf{Type} \\
\text{NF}_\Gamma(\text{El}(k)) &\equiv \text{El}(\text{NF}_\Gamma(k)) \\
\text{NF}_\Gamma((x : K)K') &\equiv (x : \text{NF}_\Gamma(K))\text{NF}_{\Gamma, x:K}(K') \\
\\
\text{NF}_\Gamma(\langle \rangle) &\equiv \langle \rangle \\
\text{NF}_\Gamma(\Delta, x : K) &\equiv \text{NF}_\Gamma(\Delta), x : \text{NF}_{\Gamma, \Delta}(K) \\
\\
\text{NF}(\Gamma \text{ valid}) &= \{\text{NF}_\langle \rangle(\Gamma) \text{ valid}\} \\
\text{NF}(\Gamma \vdash K \text{ kind}) &= (\text{NF}_\langle \rangle(\Gamma) \Vdash \text{NF}_\Gamma(K) \text{ kind}) \\
\text{NF}(\Gamma \vdash K = K') &= (\text{NF}_\langle \rangle(\Gamma) \Vdash \text{NF}_\Gamma(K) = \text{NF}_\Gamma(K')) \\
\text{NF}(\Gamma \vdash k : K) &= (\text{NF}_\langle \rangle(\Gamma) \Vdash \text{NF}_\Gamma(k) : \text{NF}_\Gamma(K)) \\
\text{NF}(\Gamma \vdash k = k' : K) &= (\text{NF}_\langle \rangle(\Gamma) \Vdash \text{NF}_\Gamma(k) = \text{NF}_\Gamma(k') : \text{NF}_\Gamma(K))
\end{aligned}$$

Given a type theory specification \mathcal{T} in LF, we define the type theory specification $\text{NF}(\mathcal{T})$ in TF as follows.

- For each declaration $c : K$ in \mathcal{T} , the declaration $c : \text{NF}_\langle \rangle(K)$ is in $\text{NF}(\mathcal{T})$.
- For each declaration $(\Delta)(k = k' : K)$ in \mathcal{T} , the declaration $(\text{NF}_\langle \rangle(\Delta))(\text{NF}_\Delta(k) = \text{NF}_\Delta(k') : \text{NF}_\Delta(K))$ is in $\text{NF}(\mathcal{T})$.

The following results ensure that this translation is well-behaved and sound.

Theorem 5.4

1. Let $\text{Ar}(K) \equiv \alpha$. If $\text{NF}_\Gamma(K)$ is defined, then it is an α -ary kind.
2. Let $\Gamma \subseteq \Delta$. If $\text{NF}_\Gamma(X)$ is defined, then $\text{NF}_\Delta(X)$ is defined, and

$$\text{NF}_\Delta(X) \equiv \text{NF}_\Gamma(X) \text{ .}$$

3. Suppose $\text{Ar}_\Gamma(k) \equiv \text{Ar}(K)$. Let X be an LF-object, kind or context. If $\text{NF}_\Gamma(k)$ and $\text{NF}_{\Gamma, x:K, \Delta}(X)$ are defined, then $\text{NF}_{\Gamma, [k/x]\Delta}([k/x]X)$ is defined, and

$$\text{NF}_{\Gamma, [k/x]\Delta}([k/x]X) \equiv \{\text{NF}_\Gamma(k)/x\}\text{NF}_{\Gamma, x:K, \Delta}(X) \text{ .}$$

4. Let T be a type-theory specification in LF, and suppose $\text{NF}(T)$ is an n -good specification in TF. If the judgement \mathcal{J} is derivable in LF and has context of order $\leq n$, then $\text{NF}(\mathcal{J})$ is defined and derivable in TF.

PROOF. The first three parts are easily proven by an induction on K and X respectively.

The fourth part is proven by induction on the derivation of \mathcal{J} . Most cases are straightforward, making use of the results proven in Section 3. We give here the details for the rule (beta).

$$\text{(beta)} \frac{\Gamma, x : K \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma \vdash ([x : K]k')k = [k/x]k' : [k/x]K'}$$

By the induction hypothesis,

$$\begin{array}{ll} \text{NF}_{\Diamond}(\Gamma), x : \text{NF}_{\Gamma}(K) & \Vdash \text{NF}_{\Gamma, x:K}(k') : \text{NF}_{\Gamma, x:K}(K'); \\ \text{NF}_{\Diamond}(\Gamma) & \Vdash \text{NF}_{\Gamma}(k) : \text{NF}_{\Gamma}(K) . \end{array}$$

Now,

$$\begin{aligned} \text{NF}_{\Gamma}([x : K]k')k &\equiv ([x]\text{NF}_{\Gamma, x:K}(k')) \bullet \text{NF}_{\Gamma}(k) \\ &\equiv \{\text{NF}_{\Gamma}(k)/x\}\text{NF}_{\Gamma, x:K}(k') \\ &\equiv \text{NF}_{\Gamma}([k/x]k') \end{aligned} \quad (\text{part 3})$$

The Cut rule and (ref) give us

$$\begin{aligned} \text{NF}_{\Diamond}(\Gamma) &\Vdash \{\text{NF}_{\Gamma}(k)/x\}\text{NF}_{\Gamma, x:K}(k') = \{\text{NF}_{\Gamma}(k)/x\}\text{NF}_{\Gamma, x:K}(k') \\ &: \{\text{NF}_{\Gamma}(k)/x\}\text{NF}_{\Gamma, x:K}(K') \end{aligned}$$

and, by part 3, this is the same judgement as

$$\text{NF}_{\Diamond}(\Gamma) \Vdash \text{NF}_{\Gamma}([x : K]k')k = \text{NF}_{\Gamma}([k/x]k') : \text{NF}_{\Gamma}([k/x]K') .$$

The translations we have established between our three systems are shown in Figure 2. The triangles in this diagram commute in the sense given by the following theorem.

Theorem 5.5 *Let T be a type theory specification in LF , and suppose $\text{NF}(T)$ is an orderable n -good type theory specification in TF .*

1. *If $\Gamma \vdash k : K$ in LF , then*

$$\Gamma \vdash k = \text{lift} \left(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}^{\text{NF}_{\Gamma}(K)}(\text{NF}_{\Gamma}(k)) \right) : K .$$

Similar results hold for kinds and contexts.

2. *If $\Gamma \vdash M : T$ in TF , then*

$$M \equiv \text{NF}_{\text{lift}(\mathcal{L}_{\Diamond}(\Gamma))}(\text{lift}(\mathcal{L}_{\Gamma}(M))) .$$

Similar results hold for kinds and contexts.

3. If $\Gamma \vdash M : T$ in TF_k , then

$$\Gamma \vdash M = \mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}(\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(M))) : T .$$

Similar results hold for kinds and contexts.

PROOF.

1. We prove the statement:

If $\Gamma \vdash k : K$ and $\Vdash \Gamma = \Delta$ in LF, then

$$\Gamma \vdash k = \text{lift}(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}^{\text{NF}_{\Gamma}(K)}(\text{NF}_{\Delta}(k))) : K .$$

We prove the statements simultaneously with similar statements for kinds and contexts by induction on size. We give here the details for the case where k is an abstraction.

Let $k \equiv [x : K_0]k'$, and $K \equiv (x : K_1)K_2$. By Generation, we have

$$\Gamma \vdash K_0 = K_1, \quad \Gamma, x : K_1 \vdash k' : K_2 .$$

Now,

$$\begin{aligned} & \text{lift} \left(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}^{\text{NF}_{\Gamma}(K)}(\text{NF}_{\Delta}(k)) \right) \\ & \equiv \text{lift} \left(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}^{(x:\text{NF}_{\Gamma}(K_1))\text{NF}_{\Gamma,x:K_1}(K_2)}([x]\text{NF}_{\Delta,x:K_0}(k')) \right) \\ & \equiv \text{lift} \left([x : \mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}(\text{NF}_{\Gamma}(K_1))] \mathcal{L}_{\text{NF}_{\Diamond}(\Gamma,x:K_1)}^{\text{NF}_{\Gamma,x:K_1}(K_2)}(\text{NF}_{\Delta,x:K_0}(k')) \right) \\ & \equiv [x : \text{lift}(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}(\text{NF}_{\Gamma}(K_1)))] \text{lift} \left(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma,x:K_1)}^{\text{NF}_{\Gamma,x:K_1}(K_2)}(\text{NF}_{\Delta,x:K_0}(k')) \right) \end{aligned}$$

Now, the induction hypothesis gives the two judgements

$$\begin{aligned} \Gamma & \vdash \text{lift}(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma)}(\text{NF}_{\Gamma}(K_1))) = K_1 \\ \Gamma, x : K_1 & \vdash \text{lift} \left(\mathcal{L}_{\text{NF}_{\Diamond}(\Gamma,x:K_1)}^{\text{NF}_{\Gamma,x:K_1}(K_2)}(\text{NF}_{\Delta,x:K_0}(k')) \right) = k' : K_2 \end{aligned}$$

from which the result follows.

2. The proof is by induction on the object M . Let $M \equiv z[\vec{F}]$, and let z have kind $(\Delta)T$ relative to Γ . Then

$$\begin{aligned} \text{NF}_{\text{lift}(\mathcal{L}_{\Diamond}(\Gamma))}(\text{lift}(\mathcal{L}_{\Gamma}(M))) & \equiv \text{NF}_{\text{lift}(\mathcal{L}_{\Diamond}(\Gamma))}(\text{lift}(z[\mathcal{L}_{\Diamond}^{\Delta}(\vec{F})])) \\ & \equiv \text{NF}_{\text{lift}(\mathcal{L}_{\Diamond}(\Gamma))}(z[\text{lift}(\mathcal{L}_{\Diamond}^{\Delta}(\vec{F}))]) \\ & \equiv z^{\eta} \bullet \text{NF}_{\text{lift}(\mathcal{L}_{\Diamond}(\Gamma))}(\text{lift}(\mathcal{L}_{\Diamond}^{\Delta}(\vec{F}))) \\ & \equiv z[\text{NF}_{\text{lift}(\mathcal{L}_{\Diamond}(\Gamma))}(\text{lift}(\mathcal{L}_{\Diamond}^{\Delta}(\vec{F})))] \\ & \equiv z[\vec{F}] \quad (\text{i.h.}) \end{aligned}$$

3. The proof is by induction on the object M . Let $M \equiv x[\vec{F}]$, and let

$$\Gamma \equiv \Gamma_1, x : (\Delta)S, \Gamma_2 \quad .$$

Then

$$\begin{aligned} & \mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}(\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(M))) \\ & \equiv \mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}(\text{NF}_{\text{lift}(\Gamma)}(x\text{lift}(\vec{F}))) \\ & \equiv \mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}(x^\eta \bullet \text{NF}_{\text{lift}(\Gamma)}(\text{lift}(\vec{F}))) \\ & \equiv \mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}(x[\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(\vec{F}))]) \\ & \equiv x \left[\mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}^{\text{NF}_{\text{lift}(\Gamma_1)}(\text{lift}(\Delta))}(\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(\vec{F}))) \right] \\ & \equiv x \left[\mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}^{\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(\Delta))}(\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(\vec{F}))) \right] \end{aligned}$$

Now, by Generation, $\Gamma \Vdash \vec{F} :: \Delta$ and $\Gamma \Vdash \{\vec{F}/\Delta\}S = T$. Hence, the induction hypothesis gives

$$\Gamma \Vdash \vec{F} = \mathcal{L}_{\text{NF}_{\Diamond}(\text{lift}(\Gamma))}^{\text{NF}_{\text{lift}(\Gamma)}(\text{lift}(\Delta))}(\text{NF}_{\text{lift}(\Gamma)}(\vec{F})) :: \Delta$$

from which the result follows.

5.4. Lifting Results

Suppose we wish to establish a property of a framework, or of an object theory in a traditional framework F . It is often the case that the property is more easily proven for a lambda-free framework L . The result can then be ‘lifted’ to F ; that is, we can derive the result for F easily from L , together with the properties of the translations between L and F .

In Luo and Adams [13], we were working with a type theory declared in LF: an extension of the type theory UTT [6] with some new reduction rules. It was found to be necessary to prove that type constructors are injective; that is, whenever $T : (K)\mathbf{Type}$ and $TA = TB$, then $A = B$. We were not able to find a way to prove this result in LF directly; the obvious method requires using the Church-Rosser property for the new reduction relation, which is not known to hold. However, the corresponding result in TF is almost trivial, and so we made use of this fact and lifted the result from TF to LF. As an illustration of the process of lifting results, we repeat the details here.

We seek to prove:

Theorem 5.6 (Injectivity of Type Constructors) *Let \mathcal{S} be a type theory specification in LF that has the property: for every equation declaration $(\Delta)(M = N : T)$ in \mathcal{S} , T has the form $\text{El}(A)$ (that is, there are no equation declarations of the form $(\Delta)(M = N : \mathbf{Type})$). Further, suppose $\text{NF}(\mathcal{S})$ is an orderable n -good specification in TF. Let $c : (\Theta)\mathbf{Type}$ be a constant declaration in \mathcal{S} . Then the following rule of deduction is admissible:*

$$\frac{\Gamma \vdash c\vec{A} = c\vec{B} : \mathbf{Type}}{\Gamma \Vdash \vec{A} = \vec{B} :: \Theta}$$

where Γ has order $\leq n$.

The corresponding result for TF is fairly easy to prove:

Theorem 5.7 *Let \mathcal{S} be a type theory specification in TF that has the property: for every equation declaration $(\Delta)(M = N : T)$ in \mathcal{S} , T has the form $\text{El}(A)$. Let $c : (\Theta)\mathbf{Type}$ be a constant declaration in \mathcal{S} . Then the following rule of deduction is admissible:*

$$\frac{\Gamma \vdash c\vec{F} = c\vec{G} : \mathbf{Type}}{\Gamma \Vdash \vec{F} = \vec{G} :: \Theta}$$

PROOF. We shall prove the following statement.

If $\Gamma \vdash c\vec{F} = X : \mathbf{Type}$ or $\Gamma \vdash X = c\vec{F} : \mathbf{Type}$ is derivable, then X has the form $c\vec{G}$, and $\Gamma \Vdash \vec{F} = \vec{G} :: \Theta$.

The proof is by induction on the derivation of the premise. Note that the last step in this derivation cannot be the use of an equation from \mathcal{S} . All cases are straightforward.

The result can now be ‘lifted’ to LF. We omit the sub- and superscripts on NF and \mathcal{L} in the following proof.

Proof of Theorem 5.6. Let \mathcal{S} satisfy the hypotheses of the theorem. Suppose $\Gamma \vdash c\vec{A} = c\vec{B} : \mathbf{Type}$ is derivable in LF under \mathcal{S} . By Theorem 5.4,

$$\text{NF}(\Gamma) \vdash c\text{NF}(\vec{A}) = c\text{NF}(\vec{B}) : \mathbf{Type}$$

is derivable in TF under $\text{NF}(\mathcal{S})$. We note also that $\text{NF}(\mathcal{S})$ satisfies the hypotheses of Theorem 5.7. Therefore,

$$\begin{aligned} \text{NF}(\Gamma) & \quad \Vdash_{TF} \quad \text{NF}(\vec{A}) = \text{NF}(\vec{B}) :: \text{NF}(\Theta) & \text{(Theorem 5.7)} \\ \therefore \mathcal{L}(\text{NF}(\Gamma)) & \quad \Vdash_{TF} \quad \mathcal{L}(\text{NF}(\vec{A})) = \mathcal{L}(\text{NF}(\vec{B})) :: \mathcal{L}(\text{NF}(\Theta)) & \text{(Theorem 4.7)} \\ \therefore \text{lift}(\mathcal{L}(\text{NF}(\Gamma))) & \quad \Vdash_{LF} \quad \text{lift}(\mathcal{L}(\text{NF}(\vec{A}))) = \text{lift}(\mathcal{L}(\text{NF}(\vec{B}))) :: \text{lift}(\mathcal{L}(\text{NF}(\Theta))) & \text{(Theorem 5.2)} \end{aligned}$$

We also have, by Theorem 5.5,

$$\begin{aligned} & \Vdash_{LF} \Gamma = \text{lift}(\mathcal{L}(\text{NF}(\Gamma))) \\ \Gamma & \Vdash_{LF} \vec{A} = \text{lift}(\mathcal{L}(\text{NF}(\vec{A}))) :: \Theta \\ \Gamma & \Vdash_{LF} \vec{B} = \text{lift}(\mathcal{L}(\text{NF}(\vec{B}))) :: \Theta \\ \Gamma & \Vdash_{LF} \Theta = \text{lift}(\mathcal{L}(\text{NF}(\Theta))) \end{aligned}$$

It follows that

$$\Gamma \Vdash_{LF} \vec{A} = \vec{B} :: \Theta$$

as required.

In contrast, the author has been unable to find a direct proof of this result in LF.

Here is a second example of how a result may be lifted from TF to LF. Let \mathcal{T} be a type theory specification in LF. Roughly, we shall show that, if $\text{NF}(\mathcal{T})$ is strongly normalising in TF, then \mathcal{T} is strongly normalising in LF.

More strictly, assume we have declared \mathcal{T} in LF and $\text{NF}(\mathcal{T})$ in TF. Suppose $\text{NF}(\mathcal{T})$ is orderable and 1-good. Let \rightarrow_R be a reduction relation on the objects of LF, and let $\rightarrow_{R\beta\eta}$ be the union of \rightarrow_R and framework-level β - and η -reduction:

$$\begin{aligned} ([x : K]k)k' & \rightarrow_{\beta} [k'/x]k \\ [x : K]kx & \rightarrow_{\eta} k . \end{aligned}$$

Define the relation \triangleright on the objects of TF as follows:

$$\begin{aligned} M \triangleright N & \text{ if and only if there exist LF-objects } a, b \text{ such that } \text{NF}(a) = M, \\ & \text{NF}(b) = N, \text{ and } M \rightarrow_R N. \end{aligned}$$

Then we have

Theorem 5.8 *Suppose that every object typable in TF is strongly \triangleright -normalising. Then every object typable in LF is strongly $\rightarrow_{R\beta\eta}$ -normalising.*

PROOF. Suppose $\Gamma \vdash a : A$ and

$$a \rightarrow_{R\beta\eta} a_1 \rightarrow_{R\beta\eta} a_2 \rightarrow_{R\beta\eta} \cdots \quad (8)$$

is an infinite \rightarrow_R -reduction sequence starting with a . Then $\text{NF}(\Gamma) \vdash_{TF} \text{NF}(a) : \text{NF}(A)$, so $\text{NF}(a)$ is strongly \triangleright -normalisable.

Now, if $a_n \rightarrow_R a_{n+1}$, then $\text{NF}(a_n) \triangleright \text{NF}(a_{n+1})$; and if $a_n \rightarrow_{\beta\eta} a_{n+1}$, then $\text{NF}(a_n) \equiv \text{NF}(a_{n+1})$. So we have

$$\text{NF}(a) \triangleright \text{NF}(a_1) \triangleright \text{NF}(a_2) \triangleright \cdots .$$

This sequence cannot contain an infinite number of \triangleright -reductions; therefore, there must be some n such that

$$\text{NF}(a_n) \equiv \text{NF}(a_{n+1}) \equiv \text{NF}(a_{n+2}) \equiv \cdots$$

and hence

$$a_n \rightarrow_{\beta\eta} a_{n+1} \rightarrow_{\beta\eta} a_{n+2} \rightarrow_{\beta\eta} \cdots$$

This contradicts the fact that LF is strongly $\rightarrow_{\beta\eta}$ -normalising.

It is often easier to prove that $\text{NF}(\mathcal{T})$ is strongly \triangleright -normalising than that \mathcal{T} is strongly $\rightarrow_{R\beta\eta}$ -normalising, because we do not have to consider how \rightarrow_R and framework-level β - and η -reduction interact.

We have made use in this proof of the fact that LF is strongly $\beta\eta$ -normalising under an arbitrary type theory specification. This is not difficult to prove, but, to the best of the author's knowledge, a proof has not yet been published, and so we present one in Appendix C.

6. Related Work

Several lambda-free logical frameworks have appeared, independently, since the publication of Adams [5].

6.1. The Canonical Logical Framework

The *Canonical Logical Framework* (Canonical LF) [7, 8] is a subsystem of the Edinburgh Logical Framework (ELF) that deals only with objects in β -normal, η -long forms. This framework uses an operation of *hereditary substitution* $[M/x]_\alpha^m N$ which behaves similarly to TF's instantiation. Their operation must be given a simple type α , which plays a similar role to the arity in TF.

The Canonical LF is essentially the same system as the following subsystem of TF_k . Let us say that a product kind $(x_1 : K_1, \dots, x_n : K_n)T$ is *small* iff the symbol **Type** does not occur in it, and *large* otherwise. We impose the following restrictions on TF:

- every variable that appears in a judgement or constant declaration must have a small kind;
- no equation declarations may be made.

This subsystem was the system named $\mathbf{SPar}(\omega)^-$ in [5]. We can prove that TF_k is conservative over this subsystem in a very strong sense:

Theorem 6.1 *Let \mathcal{T} be a type theory specification containing no equation declarations, such that every variable in a constant declaration has a small kind. Let \mathcal{J} be a judgement in which every variable has a small kind. If \mathcal{J} is derivable in TF_k under \mathcal{T} , then \mathcal{J} is derivable under \mathcal{T} in $\mathbf{SPar}(\omega)^-$. In fact, every derivation of \mathcal{J} in TF_k is a derivation of \mathcal{J} in $\mathbf{SPar}(\omega)^-$.*

Canonical LF	$\mathbf{SPar}(\omega)^-$
Kinds	Product kinds of the form $(\Delta)\mathbf{Type}$
Canonical Type Families	Product kinds of the form $(\Delta)\mathbf{El}(A)$
Atomic Type Families	Objects of kind \mathbf{Type}
Canonical Terms	Abstractions of small kind
Atomic Terms	Objects of small kind

Table 1: Correspondence between the syntactic categories of Canonical LF and $\mathbf{SPar}(\omega)^-$.

PROOF. By inspection of the rules of \mathbf{TF}_k , we see the following two facts.

1. If a variable of large kind occurs in a derivable judgement, then a variable of large kind occurs in the context of that judgement.
2. If a variable of large kind occurs in the context of a judgement at some point in a derivation, then a variable of large kind occurs in the context of every judgement below that point.

Therefore, if the conclusion contains no variable with large kind, then no variable with large kind occurs anywhere in the derivation, and the derivation is valid in $\mathbf{SPar}(\omega)^-$.

There is a close correspondence between Canonical LF and $\mathbf{SPar}(\omega)^-$. It is possible to define a bijective translation between Canonical LF and $\mathbf{SPar}(\omega)^-$ that maps each class of entity in the left-hand column of Table 1 to the corresponding class of entity in the right-hand column.

The embedding of TF in LF given in this paper can be adapted in a straightforward way to provide an embedding of Canonical LF in ELF. This embedding proves that the two systems are equivalent; that is, the derivable judgements of Canonical LF are exactly the derivable judgements of ELF that are in β -normal, η -long form. To the best of the author's knowledge, a proof of this fact has not yet been published. For further details, we refer to Adams [5], where an explicit embedding of $\mathbf{SPar}(\omega)^-$ in ELF is defined.

6.2. DMBEL

Plotkin has produced several 'algebraic frameworks' for logics and type theories, including DMBEL (Dependent Multi-Sorted Binding Equational Logic) [9, 10]. This is a framework that allows the declaration of theories involving second-order constants, and equations between objects. It is intended to be used for studying the theory of the syntax and semantics of logic and programming languages. The framework DMBEL uses operations of *first-order substitution* and *second-order substitution*, which are similar to TF's operation of instantiation $\{M/x\}N$ restricted to the cases where x is of order 0 or 1 respectively.

The framework DMBEL is essentially the same as the subsystem of \mathbf{TF}_k obtained by imposing the following restriction:

- In every constant declaration, equation declaration and judgement, every variable that appears must have a small kind of order 0 or 1.

DMBEL	SPar (2)
Type constant	constant of kind $(\Delta)\mathbf{Type}$, where Δ is small and of order ≤ 2
Term constant	constant of kind $(\Delta)\mathbf{El}(A)$, where Δ is small and of order ≤ 2
Term variable	variable of small kind and order 0
Abstraction variable	variable of small kind and order ≤ 1
Type	object of kind Type
Abstraction type	small product kind of order ≤ 1
Term	object of kind $\mathbf{El}(A)$
Abstraction term	abstraction of small kind and order ≤ 1
Context	context of order ≤ 1
Abstraction context	context of order ≤ 2
Signature	constant declarations in a specification

Table 2: Correspondence between the syntactic categories of DMBEL and **SPar**(2).

It follows that every constant that is declared must have order at most 2. This subsystem was named **SPar**(2) in Adams [5]. It can be proven that \mathbf{TF}_k is conservative over this subsystem:

Theorem 6.2 *Let \mathcal{T} be a specification in **SPar**(2), and let \mathcal{J} be a judgement in which every variable has a small kind of order 0 or 1. Then any derivation of \mathcal{J} under \mathcal{T} in \mathbf{TF}_k is a derivation of \mathcal{J} under \mathcal{T} in **SPar**(2).*

PROOF. By inspection of the rules of \mathbf{TF}_k , we see the following four facts.

1. If a variable of large kind occurs in a derivable judgement, then a variable of large kind occurs in the context of that judgement.
2. If a variable of order > 1 occurs in a derivable judgement, then a variable of order > 1 occurs in the context of that judgement.
3. If a variable of large kind occurs in the context of a judgement at some point in a derivation, then a variable of large kind occurs in the context of every judgement below that point.
4. If a variable of order > 1 occurs in the context of a judgement at some point in a derivation, then a variable of order > 1 occurs in the context of every judgement below that point.

Therefore, if the conclusion contains no variable with large kind, and no variable of order > 1 , then no variable with large kind or of order > 1 occurs anywhere in the derivation, and the derivation is valid in **SPar**(2).

There is a close correspondence between DMBEL and **SPar**(2). It is possible to define a bijective translation between DMBEL and **SPar**(2) that maps each class of entity in the left-hand column of Table 2 to the corresponding class of entity in the right-hand column.

The results in this paper thus show that the properties Cut, Functionality, Equation Validity and Context Conversion hold for DMBEL, and that DMBEL

can be conservatively embedded in LF. Further, if we remove equation declarations from DMBEL, then the resulting system can be conservatively embedded in both Canonical LF and ELF.

6.3. PAL^+

The phrase ‘lambda-free logical framework’ was originally coined to describe the framework PAL^+ [1]. This framework does not use lambda-abstraction, instead taking parametrisation and local definition as primitive. PAL^+ does not allow partial application; an n -ary function must be applied to all n arguments at once. It does still have a mechanism for forming abstractions, however; the object

$$\text{let } v[x_1 : K_1] = k : K \text{ in } v$$

in PAL^+ behaves very similarly to the lambda-abstraction $[x_1 : K_1]k$. The system TF thus involves even fewer primitive concepts than PAL^+ .

It can be proved that TF can be embedded in PAL^+ , in a similar manner to the embedding in LF. We refer to Adams [5] for the details.

7. Conclusion

We have presented the formal definition of two lambda-free logical frameworks, TF and TF_k , and proven several of their metatheoretic properties. We have defined translations between these two frameworks and the framework LF, and shown how these can be used to lift results proven in TF to LF.

The idea of a lambda-free framework has now been invented independently by several researchers, including Aczel (who invented TF), Harper and Pfenning (Canonical LF) and Plotkin (DMBEL). These frameworks are powerful in many ways. They represent object theories more faithfully than do traditional frameworks; each expression in the object theory corresponds to a unique object in the framework, rather than a $\beta\eta$ -convertibility class. Many results, such as the injectivity of type constructors or strong normalisation, are often easier to prove using a lambda-free framework than a traditional framework.

The cost is that the metatheoretic properties of a lambda-free framework are much more difficult to establish. This should be seen as a one-time cost, however; these properties need only be established for a framework once, and the framework can then be used for many object theories and the lifting of many results. We have been able to establish these properties for two large classes of object theories: those with no equation declarations, and those with only declarations of order ≤ 2 . It follows that these results hold for Canonical LF and DMBEL, as these systems are isomorphic to conservative subsystems of TF, one of which does not allow equation declarations, and one of which does not allow specifications of order > 2 .

For the future, the most immediate need is to remove this restriction on the specifications. We would dearly love to be able to prove that every orderable specification is good, as we would then be able to remove the hypotheses about the n -goodness of specifications and the order of contexts in each of the results

in this paper. Further work should also include constructing new lambda-free logical frameworks with features such as subtyping, coercive subtyping, or meta-logical reasoning, so that results can be lifted to traditional frameworks that have these features.

Acknowledgements

Thanks to Zhaohui Luo for helpful comments and proofreading. Many thanks to Randy Pollack for bringing the systems CLF and DMBEL to my attention.

References

- [1] Z. Luo, PAL+: A lambda-free logical framework, *Journal of Functional Programming* 13 (2) (2003) 317–338.
- [2] R. Harper, F. Honsell, G. Plotkin, A framework for defining logics, *Journal of the Association for Computing Machinery* 40 (1) (1993) 143–184, a preliminary version appeared in the *Proceedings of the Symposium on Logic in Computer Science*, pages 194–204, June 1987.
- [3] B. Nordström, K. Petersson, J. Smith, *Programming in Martin-Löf’s Type Theory: An Introduction*, Oxford University Press, 1990.
- [4] P. Aczel, Yet another logical framework, unpublished.
- [5] R. Adams, A modular hierarchy of logical frameworks, Ph.D. thesis, University of Manchester (2004).
- [6] Z. Luo, *Computation and Reasoning: A Type Theory for Computer Science*, no. 11 in International Series of Monographs on Computer Science, Oxford University Press, 1994.
- [7] R. Harper, D. R. Licata, Mechanizing metatheory in a logical framework, *Journal of Functional Programming* 17 (4–5) (2007) 613–673. doi:10.1017/S0956796807006430.
- [8] W. Lovas, F. Pfenning, A bidirectional refinement type system for LF, *Electron. Notes Theor. Comput. Sci.* 196 (2008) 113–128. doi:http://ds.doi.org/10.1016/j.entcs.2007.09.021.
- [9] G. Plotkin, An algebraic framework for logics and type theories, Talk given at LFMTTP’06 (August 2006).
- [10] R. Pollack, Some recent logical frameworks, Talk given at ProgLog, slides available at homepages.inf.ed.ac.uk/rpollack/export/canonicalLF_talk.pdf (February 2007).

- [11] R. Adams, A modular hierarchy of logical frameworks, in: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 - May 4, 2003, Revised Selected Papers, Vol. 3085 of LNCS, Springer, 2004, pp. 1–16.
- [12] T. Coquand, G. Huet, The calculus of constructions, Information and Computation 76 (1988) 95–120.
- [13] Z. Luo, R. Adams, Structural subtyping for inductive types with functorial equality rules, Mathematical Structures in Computer Science 18 (5) (2008) 931–972.

A. Metatheory of TF

We present here the proof of the basic metatheoretic properties of TF and TF_k . The proofs for each system are very similar; we shall work in TF for most of this section, and mark with the symbol § the changes that need to be made to obtain a proof for TF_k . These changes are all very minor. The most substantial is in Lemma A.7.

Fix a natural number n , and let \mathcal{T} be a type theory specification in TF (§ or TF_k) that is n -good. Throughout this section, we assume that every kind, context, variable, constant and abstraction that appears is of order $\leq n$.

We shall begin by proving the following two properties:

Cut. We say that the property Cut holds for a kind K if and only if, whenever $\Gamma, x : K, \Delta \vdash \mathcal{J}$, and $\Gamma \Vdash F : K$, then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}\mathcal{J}$.

Functionality. We say that the property Functionality holds for a kind K if and only if, whenever $\Gamma, x : K, \Delta \vdash M : T$ and $\Gamma \Vdash F = G : K$, then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}M = \{G/x\}M : \{F/x\}T$.

We first note:

Lemma A.1 *Let K be a kind. Suppose the properties Cut and Functionality hold for K . Then so does the following: if $\Gamma, x : K, \Delta \vdash M = N : T$ and $\Gamma \Vdash F = G : K$, then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}M = \{G/x\}N : \{F/x\}T$.*

PROOF. Suppose $\Gamma, x : K, \Delta \vdash M = N : T$ and $\Gamma \Vdash F = G : K$. Since the specification is good, we have $\Gamma \Vdash F : K$, and so Functionality gives

$$\Gamma, \{F/x\}\Delta \vdash \{F/x\}M = \{F/x\}N : \{F/x\}T .$$

The goodness of the specification also gives us $\Gamma \vdash N : T$, and so

$$\Gamma, \{F/x\}\Delta \vdash \{F/x\}N = \{G/x\}N : \{F/x\}T .$$

The result follows by (trans).

Theorem A.2 *The properties Cut and Functionality hold for every kind K .*

PROOF. The proof is by double induction, first on the kind K , second on the derivation of the judgement $\Gamma, x : K, \Delta \vdash \mathcal{J}$ or $\Gamma, x : K, \Delta \vdash M : T$.

Cut. Let $K \equiv (\Theta)T$ and $F \equiv [\text{dom } \Theta]P$. (§In TF_k , F will have the form $[\Theta']P$.) We deal here with the case where the last step in the derivation is

$$(\text{vareq}) \frac{\Gamma, x : (\Theta)T, \Delta \Vdash \vec{H}_1 = \vec{H}_2 :: \Theta}{\Gamma, x : (\Theta)T, \Delta \vdash x\vec{H}_1 = x\vec{H}_2 : \{\vec{H}_1/\Theta\}T}$$

By the induction hypothesis, we have

$$\Gamma, \{F/x\}\Delta \Vdash \{F/x\}\vec{H}_1 = \{F/x\}\vec{H}_2 :: \Theta .$$

We are also given that $\Gamma, \Theta \vdash P : T$. By Weakening and (ref),

$$\Gamma, \{F/x\}\Delta, \Theta \vdash P = P : T .$$

By repeatedly applying Lemma A.1 with each of the kinds in Θ , we have the desired conclusion

$$\Gamma, \{F/x\}\Delta \vdash \{\{F/x\}\vec{H}_1/\Theta\}P = \{\{F/x\}\vec{H}_2/\Theta\}P : \{\{F/x\}\vec{H}_1/\Theta\}T .$$

Functionality. Let $K \equiv (\Theta)T$, $F \equiv [\text{dom } \Theta]P$ and $G \equiv [\text{dom } \Theta]Q$. (§In TF_k , F will have the form $[\Theta']P$ and G the form $[\Theta'']Q$.) We deal here with the case where the last step in the derivation is

$$(\text{var}) \frac{\Gamma, x : (\Theta)T, \Delta \Vdash \vec{H} :: \Theta}{\Gamma, x : (\Theta)T, \Delta \vdash x\vec{H} : \{\vec{H}/\Theta\}T}$$

By the induction hypothesis, we have

$$\Gamma, \{F/x\}\Delta \Vdash \{F/x\}\vec{H} = \{G/x\}\vec{H} :: \Theta .$$

We are also given that

$$\Gamma, \Theta \vdash P = Q : T .$$

By repeatedly applying Lemma A.1 with each of the kinds in Θ , we have the desired conclusion

$$\Gamma, \{F/x\}\Delta \vdash \{\{F/x\}\vec{H}/\Theta\}P = \{\{G/x\}\vec{H}/\Theta\}Q : \{\{F/x\}\vec{H}/\Theta\}T .$$

We also deal with the case where the last step in the derivation is

$$(\text{conv}) \frac{\Gamma, x : K, \Delta \vdash M : \text{El}(A) \quad \Gamma, x : K, \Delta \vdash A = B : \mathbf{Type}}{\Gamma, x : K, \Delta \vdash M : \text{El}(B)}$$

We are given $\Gamma \Vdash F = G : K$; by the goodness of the specification, we also have $\Gamma \Vdash F : K$. By the induction hypothesis, we may apply Functionality to the first premise and Cut to the second to give

$$\begin{aligned} \Gamma, \{F/x\}\Delta \vdash \{F/x\}M &= \{G/x\}M : \text{El}(\{F/x\}A) \\ \Gamma, \{F/x\}\Delta \vdash \{F/x\}A &= \{F/x\}B : \mathbf{Type} . \end{aligned}$$

The result follows by (conveq).

Our next objective is to prove the following property:

Context Conversion. We say that the property Context Conversion holds for a kind K if and only if, whenever $\Gamma, x : K, \Delta \vdash \mathcal{J}$ and $\Gamma \Vdash K = K'$, then $\Gamma, x : K', \Delta \vdash \mathcal{J}$.

Once again, we need some auxiliary lemmas:

Lemma A.3 *Let K be a kind, and suppose Context Conversion holds for every kind of smaller arity than K . If $o(\Gamma) \leq n$ and $\Gamma \Vdash K = K'$ then $\Gamma \Vdash K' = K$.*

PROOF. The proof is by induction on K .

If $K \equiv \mathbf{Type}$, there is nothing to prove. If K has the form $\text{El}(A)$, we simply apply (sym).

Suppose $K \equiv (x : K_1)K_2$ and $K' \equiv (x : K'_1)K'_2$. We are given

$$\Gamma \Vdash K_1 = K'_1, \quad \Gamma, x : K_1 \Vdash K_2 = K'_2.$$

Applying Context Conversion gives $\Gamma, x : K'_1 \Vdash K_2 = K'_2$. The desired judgements

$$\Gamma \Vdash K'_1 = K_1, \quad \Gamma, x : K'_1 \Vdash K'_2 = K_2$$

follow by the induction hypothesis.

Lemma A.4 *Let Δ be a context, and suppose Context Conversion holds for every kind of smaller arity than Δ . If $\Gamma \Vdash \Delta = \Delta'$ then $\Gamma \Vdash \Delta' = \Delta$.*

PROOF. The proof is by induction on the length of Δ . The case of length 0 is trivial.

For the inductive step, let $\Delta \equiv \Delta_0, x : K$ and $\Delta' \equiv \Delta'_0, x : K'$. We are given

$$\Gamma \Vdash \Delta_0 = \Delta'_0 \quad \Gamma, \Delta_0 \Vdash K = K'.$$

By the induction hypothesis, $\Gamma \Vdash \Delta'_0 = \Delta_0$. Applying Context Conversion with each of the kinds in Δ_0 gives $\Gamma, \Delta'_0 \Vdash K = K'$, and so $\Gamma, \Delta'_0 \Vdash K' = K$ by the previous lemma.

Lemma A.5 *Suppose Context Conversion holds for every kind of lower arity than K_1 . If $\Gamma \Vdash K_1 = K_2$ and $\Gamma \Vdash K_2 = K_3$ then $\Gamma \Vdash K_1 = K_3$.*

PROOF. The proof is by induction on K_1 . The case $K_1 \equiv \mathbf{Type}$ is trivial. If K_1 has the form $\text{El}(A)$, we simply apply (trans).

Suppose $K_1 \equiv (x : J_1)L_1$, $K_2 \equiv (x : J_2)L_2$, and $K_3 \equiv (x : J_3)L_3$. We are given

$$\begin{array}{ll} \Gamma \Vdash J_1 = J_2 & \Gamma \Vdash J_2 = J_3 \\ \Gamma, x : J_1 \Vdash L_1 = L_2 & \Gamma, x : J_2 \Vdash L_2 = L_3 \end{array}$$

By Lemma A.3, we have $\Gamma \Vdash J_2 = J_1$; applying Context Conversion gives $\Gamma, x : J_1 \Vdash L_2 = L_3$. The desired judgements $\Gamma \Vdash J_1 = J_3$ and $\Gamma, x : J_1 \Vdash L_1 = L_3$ follow by the induction hypothesis.

Lemma A.6 *Suppose Context Conversion holds for every kind of lower arity than Δ_1 . If $\Gamma \Vdash \Delta_1 = \Delta_2$ and $\Gamma \Vdash \Delta_2 = \Delta_3$, then $\Gamma \Vdash \Delta_1 = \Delta_3$.*

PROOF. The proof is by induction on the length of Δ_1 . The case of length 0 is trivial.

For the inductive step, let $\Delta_1 \equiv \Theta_1, x : K_1$; $\Delta_2 \equiv \Theta_2, x : K_2$; and $\Delta_3 \equiv \Theta_3, x : K_3$. We are given

$$\begin{array}{ll} \Gamma \Vdash \Theta_1 = \Theta_2 & \Gamma \Vdash \Theta_2 = \Theta_3 \\ \Gamma, \Theta_1 \Vdash K_1 = K_2 & \Gamma, \Theta_2 \Vdash K_2 = K_3 \end{array}$$

By Lemma A.4, we have $\Gamma \Vdash \Theta_2 = \Theta_1$. Repeatedly applying Context Conversion gives us $\Gamma, \Theta_1 \Vdash K_2 = K_3$. The desired judgements

$$\Gamma \Vdash \Theta_1 = \Theta_3, \quad \Gamma, \Theta_1 \Vdash K_1 = K_3$$

follow by the induction hypothesis.

Lemma A.7 *Suppose Context Conversion holds for every kind of lower arity than K . If $\Gamma \Vdash F : K$ and $\Gamma \Vdash K = K'$, then $\Gamma \Vdash F : K'$.*

PROOF. Let $K \equiv (\Theta)T$, $K' \equiv (\Theta')T'$ and $F \equiv [\text{dom } \Theta]M$. We are given

$$\Gamma, \Theta \vdash M : T, \quad \Gamma \Vdash \Theta = \Theta', \quad \Gamma, \Theta \Vdash T = T'.$$

By (conv), we have $\Gamma, \Theta \vdash M : T'$. Applying Context Conversion with each of the kinds in Θ yields

$$\Gamma, \Theta' \vdash M : T'$$

as required.

§In TF_k , let $F \equiv [\Theta_1]M$. In addition to the above, we are given $\Gamma \Vdash \Theta = \Theta_1$ and must prove $\Gamma \Vdash \Theta' = \Theta_1$. This follows from Lemmas A.4 and A.6.

Lemma A.8 *Suppose Context Conversion holds for each of the kinds in Δ . If $\Gamma \Vdash \vec{F} :: \Delta$ and $\Gamma \Vdash \Delta = \Delta'$, then $\Gamma \Vdash \vec{F} :: \Delta'$.*

PROOF. The proof is by induction on the length of Δ and Δ' . The case of length 0 is trivial.

For the induction step, let $\Delta \equiv \Delta_0, x : K$, let $\Delta' \equiv \Delta'_0, x : K'$, and let $\vec{F} \equiv \vec{F}_0, F_1$. Then we are given

$$\begin{array}{ll} \Gamma \Vdash \vec{F}_0 :: \Delta_0 & \Gamma \Vdash F_1 : \{\vec{F}_0 / \Delta_0\}K \\ \Gamma \Vdash \Delta_0 = \Delta'_0 & \Gamma, \Delta_0 \Vdash K = K' \end{array}$$

By the induction hypothesis,

$$\Gamma \Vdash \vec{F}_0 :: \Delta'_0.$$

Applying Cut repeatedly gives

$$\Gamma \Vdash \{\vec{F}_0/\Delta_0\}K = \{\vec{F}_0/\Delta_0\}K'$$

and the desired judgement

$$\Gamma \Vdash F_1 : \{\vec{F}_0/\Delta_0\}K'$$

follows by the previous lemma.

Theorem A.9 *The property Context Conversion holds for every kind K .*

PROOF. Let $K \equiv (\Theta)T$ and $K' \equiv (\Theta')T'$, so we are given $\Gamma \Vdash \Theta = \Theta'$ and $\Gamma, \Theta \Vdash K = K'$. The proof is by double induction, first on the kind K , second on the derivation of $\Gamma, x : K, \Delta \vdash \mathcal{J}$.

We deal here with the case where the last step in the derivation is

$$(\text{var}) \frac{\Gamma, x : (\Theta)T, \Delta \Vdash \vec{F} :: \Theta}{\Gamma, x : (\Theta)T, \Delta \Vdash x\vec{F} : \{\vec{F}/\Theta\}T}$$

By the induction hypothesis, we have

$$\Gamma, x : (\Theta')T', \Delta \Vdash \vec{F} :: \Theta .$$

Applying Lemma A.8, we have

$$\begin{aligned} \Gamma, x : (\Theta')T', \Delta &\Vdash \vec{F} :: \Theta' \\ \therefore \Gamma, x : (\Theta')T', \Delta &\Vdash x\vec{F} : \{\vec{F}/\Theta\}T' \end{aligned} \quad (\text{var})$$

Applying Cut yields

$$\Gamma, x : (\Theta')T', \Delta \vdash \{\vec{F}/\Theta\}T = \{\vec{F}/\Theta\}T'$$

and the result follows by (sym) and (conv).

The case where the last step is (vareq) is similar, and the other cases are all straightforward.

This completes the proof of Theorem 3.8.

Note. The assumption of n -goodness is essential for this proof. To remove the need for it, one suggestion would be to add the following as primitive rules of TF:

$$(\text{Leq}) \frac{\Gamma \vdash M = N : T}{\Gamma \vdash M : T} \quad (\text{Req}) \frac{\Gamma \vdash M = N : T}{\Gamma \vdash N : T}$$

This would not work, however. The proof of Theorem A.2 would then fail, as we would not be able to complete the inductive step for the proof of Functionality in the case that the last step in the derivation is the rule (Req).

B. 2-good Specifications

Our aim in this section is to show that, if \mathcal{T} is an orderable type theory specification in which every declaration is of order ≤ 2 , then \mathcal{T} is 2-good. In order to prove this, we must prove four properties hold simultaneously. The following proof holds whether we are working in TF or TF_k.

Theorem B.1 *Suppose \mathcal{T} is an orderable specification, and every declaration in \mathcal{T} has order ≤ 2 . Then:*

1. *Whenever $\Gamma \vdash M = N : T$ and Γ has order ≤ 2 then $\Gamma \vdash M : T$ and $\Gamma \vdash N : T$.*
2. *Whenever $\Gamma, x : K, \Delta \vdash \mathcal{J}$, $\Gamma \Vdash F : K$ and $\Gamma, x : K, \Delta$ is of order ≤ 2 , then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}\mathcal{J}$.*
3. *Whenever $\Gamma, x : K, \Delta \vdash M : T$, $\Gamma \Vdash F = G : K$ and $\Gamma, x : K, \Delta$ is of order ≤ 2 , then $\Gamma, \{F/x\}\Delta \vdash \{F/x\}M = \{G/x\}M : \{F/x\}T$.*
4. *Whenever $\Gamma, x : K, \Delta \vdash \mathcal{J}$, $\Gamma \Vdash K = K'$ and $\Gamma, x : K, \Delta$ is of order ≤ 2 , then $\Gamma, x : K', \Delta \vdash \mathcal{J}$.*

PROOF. By the orderability of \mathcal{T} , we may replace the rules (const), (const_eq) and (eq) with the following rules without changing the set of derivable judgements. For each constant declaration $c : (\Delta)T$,

$$(\text{const}') \frac{\Gamma \Vdash \vec{F} :: \Delta \quad \Delta \Vdash T \text{ kind}}{\Gamma \vdash c\vec{F} : \{\vec{F}/\Delta\}T} \quad (\text{const_eq}') \frac{\Gamma \Vdash \vec{F} = \vec{G} :: \Delta \quad \Delta \vdash T \text{ kind}}{\Gamma \vdash c\vec{F} = c\vec{G} : \{\vec{F}/\Delta\}T}$$

For each equation declaration $(\Delta)(M = N : T)$,

$$(\text{eq}') \frac{\Gamma \Vdash \vec{F} :: \Delta \quad \Delta \vdash M : T \quad \Delta \vdash N : T}{\Gamma \vdash \{\vec{F}/\Delta\}M = \{\vec{F}/\Delta\}N : \{\vec{F}/\Delta\}T}$$

Given a finite sequence of declarations s , let us write $\Gamma \vdash_s \mathcal{J}$ to mean that there exists a derivation of the judgement $\Gamma \vdash \mathcal{J}$ such that, for every branch in the derivation, the declarations used at the (const), (const_eq) and (eq) nodes, taken in order from leaf to root, form a subsequence of s . For defined judgement forms, we write (e.g.) $\Gamma \Vdash_s (x : \text{El}(A))\text{El}(B) = (x : \text{El}(A'))\text{El}(B')$ to mean $\Gamma \vdash_s A = A' : \mathbf{Type}$ and $\Gamma, x : A \vdash_s B = B'$ both hold.

We write $s \sqsubset t$ to denote that s is a proper initial segment of t . We write $\Gamma \vdash_{\sqsubset s} \mathcal{J}$ to denote that there exists $t \sqsubset s$ such that $\Gamma \vdash_t \mathcal{J}$.

Define the *order* of a sequence s by

$$o(s) = \max\{o(\delta) \mid \delta \in s\} .$$

We define the following properties for natural numbers m, n with $n < m$ and sequences s .

- $\text{CUT}(m, n, s)$ is the statement: whenever $\Gamma, x : K, \Delta$ has order m , K has order n , and $\Gamma, x : K, \Delta \vdash_s \mathcal{J}$ and $\Gamma \Vdash_s F : K$, then $\Gamma, \{F/x\}\Delta \vdash_s \{F/x\}\mathcal{J}$.
- $\text{FUNC}(m, n, s)$ is the statement: whenever $\Gamma, x : K, \Delta$ has order m , K has order n , and $\Gamma, x : K, \Delta \vdash_s M : T$ and $\Gamma \Vdash_s F = G : K$, then $\Gamma, \{F/x\}\Delta \vdash_s \{F/x\}M = \{G/x\}M : \{F/x\}T$.
- $\text{CC}(m, n, s)$ is the statement: whenever $\Gamma, x : K, \Delta$ has order m , K has order n , and $\Gamma, x : K, \Delta \vdash_s \mathcal{J}$ and $\Gamma \Vdash_s K = K'$, then $\Gamma, x : K', \Delta \vdash_s \mathcal{J}$.
- $\text{EQVAL}(m, s)$ is the statement: whenever Γ has order m and $\Gamma \vdash_s M = N : T$, then $\Gamma \vdash_s M : T$ and $\Gamma \vdash_s N : T$. (EQVAL stands for ‘equation validity’.)
- $\text{FUNCEQ}(m, n, s)$ is the statement: whenever $\Gamma, x : K, \Delta$ has order m , K has order n , and $\Gamma, x : K, \Delta \vdash_s M = N : T$ and $\Gamma \Vdash_s F = G : K$, then $\Gamma, \{F/x\}\Delta \vdash_s \{F/x\}M = \{G/x\}N : \{F/x\}T$.
- $\text{GFUNC}(m, n, s)$ is the statement: whenever $\Gamma, x : K, \Delta$ has order m , K has order n , and $\Gamma, x : K, \Delta \vdash_s M : T$, $\Gamma \Vdash_s F = G : K$, $\Gamma \Vdash_s F : K$ and $\Gamma \Vdash_s G : K$, then $\Gamma, \{F/x\}\Delta \vdash_s \{F/x\}M = \{G/x\}M : \{F/x\}T$. (GFUNC stands for ‘guarded functionality’.)

We shall employ the following abbreviations: $\text{CUT}(\leq a, < b, s)$, for example, shall mean that $\text{CUT}(m, n, s)$ holds for all $m \leq a$ and all $n < b$. Another example: $\text{CC}(m, n, \leq s)$ shall mean $\text{CUT}(m, n, t)$ holds for all $t \leq s$.

Our aim is to show $\text{EQVAL}(2, s)$ for all sequences s of declarations from \mathcal{T} .

By proofs similar to the ones in the Appendix A, we can prove the following results for all m and s :

- (1) $\text{FUNCEQ}(m, < n, s) \wedge \text{CUT}(m, < n, s) \Rightarrow \text{GFUNC}(m, n, s)$
- (2) $\text{CUT}(m, < n, s) \wedge \text{FUNCEQ}(m, < n, s) \Rightarrow \text{CUT}(m, n, s)$
- (3) $\text{CUT}(m, < n, s) \wedge \text{CC}(m, < n - 1, s) \wedge \text{EQVAL}(m, s) \Rightarrow \text{CC}(m, n, s)$
- (4) $\text{GFUNC}(m, n, s) \wedge \text{CUT}(m, n, s) \Rightarrow \text{GFUNCEQ}(m, n, s)$

The following results are trivial:

- (5) $\text{GFUNC}(m, n, s) \wedge \text{EQVAL}(m, s) \Rightarrow \text{FUNC}(m, n, s)$
- (6) $\text{GFUNCEQ}(m, n, s) \wedge \text{EQVAL}(m, s) \Rightarrow \text{FUNCEQ}(m, n, s)$

Claim.

(7) The properties

$$\begin{aligned}
& \text{GFUNCEQ}(m, < m - 1, s) \\
& \text{CC}(m, < m - 2, s) \\
& \text{GFUNCEQ}(\leq \max(m, o(s)), < o(s), \sqsubset s) \\
& \text{CC}(\leq \max(m, o(s) - 1), < o(s) - 1, \sqsubset s) \\
& \text{CUT}(\leq \max(m, o(s)), < o(s), \sqsubset s)
\end{aligned}$$

entail $\text{EQVAL}(m, s)$.

Proof. We prove that, whenever Γ has order $\leq m$ and $\Gamma \vdash_s M = N : T$, then $\Gamma \vdash_s M : T$ and $\Gamma \vdash_s N : T$, by induction on the derivation of $\Gamma \vdash_s M = N : T$.

Suppose the last step in the derivation is

$$(\text{const_eq}') \frac{\Gamma \Vdash_s \vec{F} = \vec{G} :: \Delta \quad \Delta \Vdash T \text{ kind}}{\Gamma \vdash_s c\vec{F} = c\vec{G} : \{\vec{F}/\Delta\}T}$$

where we have $(c : (\Delta)T) \in s$. Let $s = s_1, c : (\Delta)T, s_2$, where $c : (\Delta)T$ does not occur in s_2 .

The induction hypothesis gives $\Gamma \Vdash_{s_1} \vec{F} :: \Delta$, and so $\Gamma \vdash_s c\vec{F} : \{\vec{F}/\Delta\}T$ by (const).

The induction hypothesis also gives $\Gamma \vdash_{s_1} G_i : \{\vec{F}/\Delta\}K_i$, where K_i is the i th kind in Δ . By Context Validity, we also have

$$x_1 : K_1, \dots, x_{i-1} : K_{i-1} \Vdash_{s_1} K_i \text{ kind} .$$

Using $\text{GFUNCEQ}(m, < o(s), \sqsubset s)$, we have $\Gamma \Vdash_{s_1} \{\vec{F}/\Delta\}K_i = \{\vec{G}/\Delta\}K_i$, and so, using $\text{CC}(m, < o(s) - 1, \sqsubset s)$,

$$\Gamma \vdash_{s_1} G_i : \{\vec{G}/\Delta\}K_i ,$$

that is, $\Gamma \Vdash_{s_1} \vec{G} :: \Delta$. Therefore, $\Gamma \vdash_s c\vec{G} : \{\vec{G}/\Delta\}T$ by (const).

The case (vareq) is similar, using $\text{GFUNCEQ}(m, < m - 1, s)$ and $\text{CC}(m, < m - 2, s)$.

Suppose $s = s_1, (\Delta)(M = N : T), s_2$, and the last step in the derivation is

$$(\text{eq}') \frac{\Gamma \Vdash_{s_1} \vec{F} :: \Delta \quad \Delta \vdash_{s_1} M : T \quad \Delta \vdash_{s_1} N : T}{\Gamma \vdash_s \{\vec{F}/\Delta\}M = \{\vec{F}/\Delta\}N : \{\vec{F}/\Delta\}T}$$

By $\text{CUT}(\leq \max(m, o(s)), < o(s), \sqsubset s)$, we have $\Gamma \vdash_{s_1} \{\vec{F}/\Delta\}M : \{\vec{F}/\Delta\}T$ and $\Gamma \vdash_{s_1} \{\vec{F}/\Delta\}N : \{\vec{F}/\Delta\}T$.

We can now use these seven results to prove Theorem B.1. Firstly, note that (1) and (2) imply

$$\text{GFUNC}(m, 0, s) \wedge \text{CUT}(m, 0, s)$$

for every m and s . Therefore, by (4), $\text{GFUNCEQ}(m, 0, s)$ holds for every m and s .

Our goal is to prove the following:

$$\text{EQVAL}(2, s) \wedge \text{FUNC}(2, 1, s) \wedge \text{CUT}(2, 1, s) \wedge \text{CC}(2, 1, s) \quad .$$

The proof is by induction on the length of s . Suppose, as induction hypothesis,

$$\text{EQVAL}(2, \sqsubset s) \wedge \text{FUNC}(2, 1, \sqsubset s) \wedge \text{CUT}(2, 1, \sqsubset s) \wedge \text{CC}(2, 1, \sqsubset s) \quad .$$

Then the following hold:

$$\begin{array}{ll} \text{CC}(2, \leq 1, \sqsubset s) & \text{(by (3))} \\ \text{EQVAL}(2, s) & \text{(by (7))} \\ \text{FUNC}(2, 0, s) & \text{(by (5))} \\ \text{FUNCEQ}(2, 0, s) & \text{(by (6))} \\ \text{GFUNC}(2, 1, s) & \text{(by (1))} \\ \text{FUNC}(2, 1, s) & \text{(by (5))} \\ \text{CUT}(2, 1, s) & \text{(by (2))} \\ \text{GFUNCEQ}(2, 1, s) & \text{(by (4))} \\ \text{FUNCEQ}(2, 1, s) & \text{(by (6))} \end{array}$$

This completes the induction.

It does not seem possible to use the same method to prove that, if every declaration in \mathcal{T} is of order ≤ 3 , then \mathcal{T} is 3-good. As noted in the proof, we have $\text{GFUNC}(m, 0, s)$, $\text{CUT}(m, 0, s)$ and $\text{GFUNCEQ}(m, 0, s)$. It is also possible to prove directly, by an induction on derivations, that $\text{CC}(m, 0, s)$ holds for all m and s . We are then stuck: for $o(s) = 2$, we have the circle of implications

$$\begin{aligned} \text{EQVAL}(3, s) &\Rightarrow \text{FUNCEQ}(3, 0, s) \Rightarrow \text{GFUNC}(3, 1, s) \wedge \text{CUT}(3, 1, s) \\ &\Rightarrow \text{GFUNCEQ}(3, 1, s) \Rightarrow \text{EQVAL}(3, s) \end{aligned}$$

without any immediate way to prove any of these directly.

We are thus unable to prove the following statement yet, and present it here as a conjecture:

Conjecture B.2 *Every orderable type theory specification is good.*

C. The Strong Normalisability of LF

Consider the simply typed lambda-calculus (STLC), with the following grammar:

$$\begin{array}{ll} \text{Type } A & ::= * \mid A \rightarrow A \\ \text{Term } M & ::= x \mid \lambda x : A. M \mid MM \end{array}$$

We shall use the fact that every term typable in STLC is strongly $\beta\eta$ -normalising to prove that every object typable in LF is strongly $\beta\eta$ -normalising.

Define a translation $\llbracket \cdot \rrbracket$ that maps every kind of LF to a type of STLC, every object of LF to a term of STLC, and every context of LF to a context of STLC, as follows:

$$\begin{aligned}
\llbracket \mathbf{Type} \rrbracket &\equiv * \\
\llbracket \mathbf{El}(k) \rrbracket &\equiv * \\
\llbracket (x : K)K' \rrbracket &\equiv \llbracket K \rrbracket \rightarrow \llbracket K' \rrbracket \\
\llbracket [x : K]k \rrbracket &\equiv \lambda x : \llbracket K \rrbracket . \llbracket k \rrbracket \\
\llbracket kk' \rrbracket &\equiv \llbracket k \rrbracket \llbracket k' \rrbracket \\
\llbracket x_1 : K_1, \dots, x_n : K_n \rrbracket &\equiv x_1 : \llbracket K_1 \rrbracket, \dots, x_n : \llbracket K_n \rrbracket
\end{aligned}$$

The key step in this proof is to realise the following fact about this translation:

Lemma C.1 *Under an arbitrary type theory specification in LF, if $\Gamma \vdash K = K'$, then $\llbracket K \rrbracket \equiv \llbracket K' \rrbracket$.*

PROOF. The proof is a simple induction on derivations.

Using this lemma, we can establish the following:

Lemma C.2 *Suppose $\Gamma \vdash k : K$. Let c_1, \dots, c_m be the constants that occur in k , and let them be declared with kinds*

$$c_1 : K_1, \dots, c_m : K_m .$$

Then

$$c_1 : \llbracket K_1 \rrbracket, \dots, c_m : \llbracket K_m \rrbracket, \llbracket \Gamma \rrbracket \vdash \llbracket k \rrbracket : \llbracket K \rrbracket$$

in STLC.

PROOF. The proof is by induction on the derivation of $\Gamma \vdash k : K$.

Lemma C.3 *If k and k' are LF-objects and $k \rightarrow_{\beta\eta} k'$, then $\llbracket k \rrbracket \rightarrow_{\beta\eta} \llbracket k' \rrbracket$.*

PROOF. We first establish the fact that

$$\llbracket [k/x]k' \rrbracket \equiv (\llbracket k \rrbracket / x) \llbracket k' \rrbracket$$

by induction on k' . Now, if $k \equiv ([x : K]k_1)k_2$ and $k' \equiv [k_2/x]k_1$, then

$$\llbracket k \rrbracket \equiv (\lambda x : \llbracket K \rrbracket . \llbracket k_1 \rrbracket) \llbracket k_2 \rrbracket \rightarrow_{\beta} \llbracket [k_2/x]k_1 \rrbracket \equiv \llbracket k' \rrbracket .$$

The other cases are similar.

These allow us to prove the theorem we want:

Theorem C.4 *Under an arbitrary type theory specification in LF, if $\Gamma \vdash k : K$, then k is strongly $\beta\eta$ -normalising.*

PROOF. Suppose $k \rightarrow_{\beta\eta} k_1 \rightarrow_{\beta\eta} k_2 \rightarrow_{\beta\eta} \dots$ is an infinite reduction sequence. By Lemma C.2, we have that $\llbracket k \rrbracket$ is typable in STLC under some context; and by Lemma C.3, we have that

$$\llbracket k \rrbracket \rightarrow_{\beta\eta} \llbracket k_1 \rrbracket \rightarrow_{\beta\eta} \llbracket k_2 \rrbracket \rightarrow_{\beta\eta} \dots$$

is an infinite reduction sequence. This contradicts the fact that STLC is strongly normalising.